ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

INGEGNERIA ELETTRONICA, TELECOMUNICAZIONI E TECNOLOGIE DELL'INFORMAZIONE

Ciclo 36

Settore Concorsuale: 09/E3 - ELETTRONICA

Settore Scientifico Disciplinare: ING-INF/01 - ELETTRONICA

ENABLING MULTI-TASKING AI-BASED PERCEPTION ON AUTONOMOUS NANO-UAVS

Presentata da: Lorenzo Lamberti

Coordinatore Dottorato

Aldo Romani

Supervisore

Luca Benini

Co-supervisori

Daniele Palossi
Francesco Conti

Esame finale anno 2024

A mio nonno Enzo.

Acknowledgements

A PhD is far more than a solitary endeavor. Thus, I used "we" instead of "I" in my thesis as I received constant help throughout, and now I would like to acknowledge some of the people who helped me along the way.

First of all, I express my deepest gratitude to my supervisor, Prof. Luca Benini, for allowing me to pursue my PhD in his amazing research group. His guidance throughout my PhD years was invaluable, and his infectious enthusiasm across a wide range of research topics continually fueled my motivation and passion.

Second, I owe a special thanks to Daniele Palossi, whose relentless support, humor, late-night calls, debugging sessions, and last-minute paper-writing assistance – not to mention his LEGO set recommendations – slowly transformed him from a great mentor into a great friend.

My appreciation extends to Francesco Conti and Manuele Rusci for their patience and invaluable guidance, especially during the early stages of my PhD, including the very traumatic experience of writing my first research paper. I'm much obliged to Prof. Davide Scaramuzza for kindly hosting me in his lab during my academic journey and to all the PhD students in his lab for being so nice and welcoming.

I am grateful to Victor Kartsch and Alfio Di Mauro for sharing their endless experience whenever I reached out. I also thank Giuseppe Tagliavini for bringing extreme joy to the lab, refilling the coffee machine, sharing food recipes with me, and, most importantly, organizing the memorable Neurosalama event every year.

My gratitude extends to all my colleagues in Bologna and Zürich, whose names are too numerous to list but are certainly not forgotten. My time at Neurolab was significantly improved by my lab mates, with whom I shared countless joyful moments and evening get-togethers. A heartfelt shout-out to my early journey companions: Luca Valente, Nazareno Bruschi, Gianmarco Ottavi, Luka Macan, Luca Bompani, my dearest friend Simone Sindaco, and Benedetta Mazzoni, who never missed taking care of ordering sushi every Friday at noon. A special mention also goes to Lorenzo Bellone, who stayed up all night with me multiple times to complete the experiments of our papers.

Reflecting on my earlier academic experiences, I am deeply grateful to those I met during my Master's Thesis in Pasadena; their passion for research was truly inspirational for me, eventually influencing my decision to pursue a PhD program. For this, I thank Filippo Mambelli and Luis Goncalves, who welcomed me into their research lab and supervised my thesis, and Joe Bortz and Spencer Buebel, my dearest friends who made me feel at home in the US.

Finally, the deepest gratitude goes to my family. My uncle Giacomo was the first to spark my interest in electronic engineering and took a keen interest in my projects. Nonna Marisa may not fully grasp what a PhD entails, but her pride is palpable. Nonno Enzo, although not here to see my journey's end, was always profoundly proud of me from the start. Above all, I owe my biggest gratitude to my mom, Silvia, and my dad, Paolo, whose unwavering love and support have been the cornerstone of my life.

Abstract

In the dynamic landscape of modern technology, artificial intelligence (AI) is rapidly advancing, enabling increasingly complex capabilities in tiny flying robots. In the most pioneering scenario, tiny AI-driven unmanned aerial vehicles (UAVs) are envisioned to achieve the same level of intelligence that is akin to biological systems, such as insects. Bees, for example, highlight the capability to pursue multiple goals concurrently and with full autonomy. The development of such sophisticated skills in miniaturized UAVs holds the potential for their effective utilization across a wide range of applications, which can significantly impact many aspects of our lives.

However, the miniaturization of UAVs carries several challenges. Nano-drones, approximately 10 cm in diameter, are at the developmental forefront. Yet, the concurrent execution of multiple intelligence tasks on nano-UAVs is still out of reach, as their limited size and payload only allow them to embed ultra-low-power (ULP) processors with stringent computational and memory constraints. This limitation sets nano-UAVs apart from the multi-tasking capabilities that biological systems inherently have.

This thesis aims to narrow the intelligence gap between tiny flying robots and insects. To achieve such an ambitious goal, this thesis enables the concurrent execution of multiple real-time AI-based perception tasks on autonomous nano-UAVs.

First, we present methodologies and software tools for automating and optimizing convolutional neural networks (CNNs) deployment on nano-UAVs while complying with their constrained ULP processors. As an example, we apply our methodology to a CNN for visual autonomous navigation, demonstrating the robustness of our pipeline in the context of an international drone competition.

Second, we thoroughly study how to minimize the CNN workload on nano-drones. With our methodology, we first study whether there are inactive neurons inside the targeted CNN, and then we introduce architecture modifications to shrink the network. We apply this methodology to a SoA visual-based autonomous navigation CNN for nano-drones, obtaining a CNN that is 50× smaller and 8.5× faster than

the baseline with no compromise on the performance metrics.

Third, leveraging the computational resources freed thanks to our CNN shrinking methodology, we enable nano-UAVs to perform multiple AI tasks in real-time by deploying a CNN for object detection in addition to the visual-based navigation one.

Combining our techniques for CNN optimization and automated deployment, ultimately integrating two CNNs on a ULP processor, we demonstrate that it is possible to overcome both the computational and memory burden imposed by ULP MCUs, allowing the simultaneous execution of multiple AI-based perception tasks on nano-UAVs. This milestone takes tiny flying robots one step closer to the high-level intelligence of tiny biological systems and their multi-tasking capabilities.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In the last decade, the ever-growing field of artificial intelligence (AI) has significantly advanced the development of smart robots, enabling them to accomplish increasingly complex and sophisticated tasks. Unmanned aerial vehicles (UAVs), namely aircraft operating without a human onboard, earned particular attention for commercial and research purposes, as the ability to fly makes them stand out for their versatility and agility. Nowadays, UAVs bring great aid when employed in a plethora of use cases [3], such as the exploration of hazardous environments [4, 5, 6], inspection of industrial facilities [7], monitoring and surveillance [8], search and rescue missions [9], precision agriculture [10], and even entertainment [11, 12]. UAVs, and robots more in general, can operate with various degrees of autonomy: remotely controlled by a human operator, automatically guided by a remote workstation, or in complete autonomy by relying solely on onboard computation and sensors. This thesis focuses on fully autonomous UAVs, meaning that the aircraft independently execute missions without relying on human intervention or any other external resources [13], such as power-unconstrained remote computers or localization systems. Operating with such a level of autonomy brings several advantages [14]: it improves reliability, as there is no risk that the noise on the transmission channel or network-dependent latency affects the transmitted data; it increases security, being immune to denial-of-service attacks on the wireless connections and preventing eavesdropping on the transmitted data; it allows for power saving, as no power consumption is dedicated to high-bandwidth radio transmissions.

Consequently, enhancing the onboard intelligence of UAVs has become crucial, as it empowers these drones to handle their missions autonomously. UAVs can be categorized based on their size and weight, as detailed in Table 1.1. On the one hand, the larger drones, namely standard and micro-sized UAVs, already demonstrate remarkable onboard intelligence, even when operating autonomously. Their large size (greater than 25cm) and weight (greater than 0.5kg) provide sufficient

payload capacity to carry powerful processors (i.e., CPUs, GPUs), like the Nvidia AGX Orin [15] (having a peak throughput of 200 TOps/s at 40 W), and sophisticated sensors [16] (i.e., high-resolution cameras, LIDARs) aboard. This enables them to perform complex tasks by running multiple artificial intelligence algorithms concurrently within real-time constraints. These UAVs have advanced to a level where their multi-tasking AI capabilities enabled them to exceed human skills: an autonomous drone set a significant milestone in the evolution of UAV intelligence by outperforming the human world champion in a drone racing competition [17].

Table 1.1. UAVs taxonomy by vehicle class-size [2].

Vehicle class	⊘ : Weight [cm:kg]	Power [W]	Onboard Device
standard-size	$\sim 50 : \geq 1$	≥ 100	Desktop
micro-size	$\sim 25 : \sim 0.5$	~ 50	Embedded
nano-size	$\sim 10 : \sim 0.01$	~ 5	MCU

However, in the most visionary scenario, we would like to bring the same level of sophisticated skills into tiny, insect-scale UAVs. The miniaturization of UAVs brings several advantages, enabling applications that are out of reach for their larger counterparts. This includes the ability to navigate in narrow spaces [18], and their light weight ensures safe operation in close-proximity with humans [19, 7], e.g., in indoor settings. Additionally, the cost-effectiveness of producing small and simple electronic components stands out as an important factor for widespread adoption. With these characteristics, tiny flying robots have the potential to become pervasive in everyday life, especially within the ever-growing Internet of Things (IoT) world, characterized by a vast network of wirelessly interconnected smart devices. Tiny UAVs are well-suited to join this network of interconnected devices by becoming the ultimate IoT node, which autonomously navigates environments while simultaneously sensing, analyzing, and interacting with their surroundings [20]. Within the IoT world, tiny UAVs can find applications in a plethora of scenarios: in household environments, e.g., acting as mobile surveillance cameras [21], industrial facilities, e.g., helping in warehouse logistics [7] or inspecting narrow tunnels/pipes [22], and in search and rescue missions, where collapsed buildings prevent the use of larger UAVs [23].

Nano-sized drones [18], with a diameter of approximately 10cm and weighing a few tens of grams, represent cutting-edge research in achieving this ambitious goal. Running AI algorithms on such constrained platforms presents numerous challenges: their compact size and limited payload restrict the size of their batteries and printed circuit boards, allowing them to carry only compact processing devices

like MicroController Units (MCUs) and low-power sensors [24]. Their total computational power budget is typically a few hundreds of mW. Despite these limitations, these tiny drones have already reached significant milestones in their autonomous navigation capabilities; they have successfully demonstrated the ability to run single convolutional neural networks (CNNs) entirely onboard, as highlighted in works like [18] and [19]. Yet, the defining factor that distinguishes the autonomy of nano-drones from their larger counterparts lies in their potential to undertake complex missions with multiple goals, which demands the execution of multiple intelligence tasks concurrently on the drone [25, 26, 12]. Such a high level of intelligence is akin to the one of any biological system, even at the insect scale. For instance, consider the remarkable capabilities of bees [27, 28]: they navigate vast distances, find and pollinate plants, communicate with their hive through intricate dances, and constantly adapt to environmental changes. This seamless integration of navigation, communication, environmental adaptation, and task execution in bees serves as a compelling model for the autonomous functionality we aim to replicate in nano-UAVs.

This thesis aims to bridge this gap by pushing the capabilities of nano-sized UAVs closer to the ones of biological systems, which are autonomous and can handle multiple concurrent high-level perception tasks that span from basic control functionality to high-level perception.

To progress towards this ambitious goal, this thesis: i) presents methodologies and software tools to streamline and automate all the deployment of vision-based CNNs on nano-drones while complying with their constrained ULP processors. ii) thoroughly studies how to minimize the CNN workload on nano-drones both in an automated and hand-crafted way, ultimately freeing up enough resources to unlock the execution of multiple AI pipelines on nano-UAVs. iii) developing and deploying multiple vision-based AI tasks running in real-time on nano-drones, specifically focusing on *autonomous navigation* and *object detection*.

Combining all of these techniques, this thesis demonstrates that it is possible to overcome both the computational and memory burden imposed by ULP MCUs and achieve the execution of multiple CNNs in real time on nano-drones. This accomplishment represents a significant milestone in the SoA for embedding intelligence in nano-sized drones, bringing these compact aerial vehicles a step closer to the sophisticated, multi-functional autonomy that tiny biological systems inherently have.

1.1 Thesis outline and contributions

An overview of the structure of this thesis is depicted in Figure 1.1. The content of each chapter is summarized in the following.

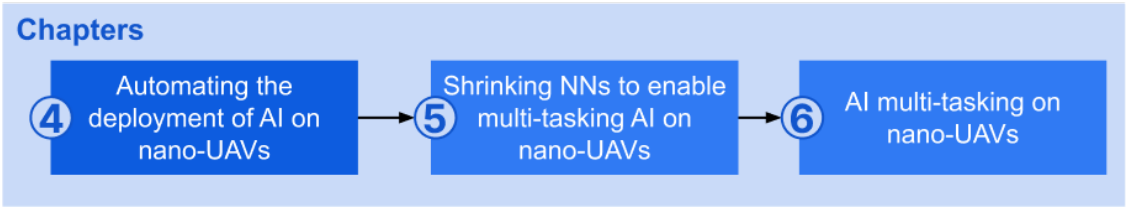


Figure 1.1. Flow of the this thesis. The Chapters' number is indicated within circle shapes.

Chapter 2 - Related works This chapter presents the state-of-the-art works related to this manuscript. First, it introduces the taxonomy of UAVs. Then, it reviews TinyML solutions to enable AI functionalities on fully autonomous nano-UAVs, including end-to-end solutions and automatic CNN deployment tools for MCUs. Furthermore, it reviews the datasets in the SoA for enabling autonomous navigation on nano-UAVs. Then, it reviews autonomous exploration strategies and visual object detection CNNs suitable for deployment on the MCUs of the nano-UAV platform. Finally, the chapter delves into the current state of the art in drone racing. This section highlights navigation through gates and visual obstacle avoidance techniques with a focus on nano-UAVs.

Chapter 3 - Background This chapter introduces the hardware background of this thesis. It describes the nano-UAV robotic platform used, including additional modules and sensors. This chapter also describes the architecture of the parallel ultra-low-power processor used to enable visual AI applications on such a constrained platform: the GAP8 MCU.

Chapter 4 - Automating the deployment of AI on MCUs. The limited computational and memory resources available aboard nano-UAVs introduce the challenge of minimizing and optimizing vision-based deep neural networks (DNNs), which require error-prone, labor-intensive iterative development flows. Conversely, this Chapter explores methodologies and software tools to streamline and automate all the deployment of a vision-based navigation CNN on an ultra-low-power (ULP) multi-core System-on-Chip (SoC), which is acting as the AI brain on a nano-UAV. We take as an example the deployment of PULP-Dronet [18], a SoA CNN for autonomous navigation of nano-UAVs, from the initial training to the final closed-loop evaluation. With our pipeline, we reduce by 2× the memory footprint of PULP-Dronet and improve its inference throughput by 1.6× by employing an 8-bit quantization scheme instead of the 16-bit original one. Last, we obtain the same prediction accuracy on the testing set while significantly improving the behavior in the field.

Furthermore, we highlight the effectiveness of our automated pipeline for deploying DNNs on MCUs in the context of a real-world drone racing competition. We developed a fully onboard deep learning approach for visual navigation and exploration, leading to a victory in the "IMAV 2022 Nanocopter AI Challenge," the first international competition for autonomous nano-drones. We achieved this result by exploiting the optimization pipeline outlined in this chapter, which adeptly addresses the stringent memory, computational, and sensor limitations inherent to the nano-drone platform used in the competition. Our approach is based on a variant of the PULP-Dronet CNN, introduced in this chapter: we leverage an end-to-end approach involving a single CNN that predicts its outputs from the raw input images only. Additionally, we take an additional step to ease the development of AI capabilities on nano-drones. We mitigate the lack of a dedicated dataset in the nano-UAV setting, which stifles the development of novel AI algorithms in this niche, by designing a new CNN trained exclusively with simulated data. We also introduce a sim-to-real mitigation strategy for real-world deployment. Our system showcased its capabilities at the IMAV'22 competition, where it ranked first among six contending teams, covering a distance of 115 meters in a 5-minute flight without experiencing any crashes. It successfully navigated dynamic obstacles, relying solely on the onboard computational resources of the nano-drone. This achievement proves the robustness and effectiveness of our automated deployment pipeline in a challenging, competitive environment. It underlines the feasibility of utilizing simulated data to enable nano-UAVs to address real-world challenges.

Chapter 5 - Shrinking NNs to enable multi-tasking AI on nano-UAVs. In Chapter 4, we demonstrated that executing individual AI tasks, such as visual-based navigation, is feasible on autonomous nano-UAVs. However, the computational and memory burden of this network is still too high to achieve multi-tasking AI perception on constrained MCUs. Therefore, this chapter aims to minimize the onboard intelligence workload needed for UAVs' autonomous navigation, freeing up enough resources to execute multi-perception intelligence tasks aboard a nano-drone. To do so, we first present a general methodology for evaluating whether deep learning models are overparametrized, i.e., determining if the same tasks could be effectively solved with smaller DNNs, specifically in terms of their size and complexity. This methodology involves analyzing overfitting behaviors and examining the sparsity of CNN's neurons. As an example, we apply this methodology to the PULP-Dronet CNN, studying the various trade-offs between the number of channels, pruning of inactive neurons, architecture modifications, and accuracy. We ultimately demonstrate the effectiveness of our method by introducing novel squeezed CNN models called Tiny-PULP-Dronets, which are up to 50× smaller and 8.5× faster than the baseline CNN when running on the same processor. Nevertheless, we show that these models maintain the regression and classification performance when validated on the testing set. These networks leave sufficient memory and

compute headroom for onboard multi-tasking intelligence, even at the nano-sized scale.

Before testing such networks in the field, we want to further improve the visual-based CNN for visual-based navigation by addressing the shortcomings of PULP-Dronet, introduced in Chapter 4. This CNN shows limited performance when trying to avoid static obstacles, therefore restricting its applicability in real-world scenarios. This limitation depends on the original CNN's dataset construction, which combines data from different robotic domains with disjoint training labels to train the two CNN outputs, i.e., the collision avoidance and steering outputs. Therefore, we introduce a dataset-collecting methodology for collecting unified collision avoidance and steering information directly on the nano-UAV, and we use this methodology to collect a new dataset of 66 k images. Second, using this dataset, we introduce end-to-end training and deployment of a new family of CNNs, called PULP-Dronet-V3, that *i*) improves the obstacle avoidance capabilities in specific scenarios, i.e., static obstacles, and *ii*) we expand the ablation study on the architectural modifications of our CNN. Finally, we present an extensive in-field validation of the resulting tiny CNN for autonomous navigation on nano-UAVs, studying how the reduced workload affects navigation accuracy. We set up a challenging scenario consisting of a narrow corridor with four static obstacles and a 180° turn, and the new tiny CNN we designed shows the ability to successfully navigate with a 100% success rate through this scenario at a maximum target speed of 0.5 m/s. At the same time, the SoA un-pruned CNN consistently fails.

As a result, the CNN that resulted from our shrinking methodology only uses a fraction of the computational resources available on nano-UAVs: we obtained a 2.9 kB model achieving 139 frame/s throughput. These results pave the way towards embedding multiple intelligence tasks on this nano-sized class of vehicles, as we can build up more intelligence on top of this optimized CNN for autonomous navigation.

Chapter 6 - AI multi-tasking on nano-UAVs. In this Chapter, we take another step forward in the nano-UAVs SoA toward a higher level of intelligence. As biological entities pursue multi-objective missions in complete autonomy, we aim to enable these nano-UAVs to execute multiple AI algorithms fully onboard. First, we enable autonomous nano-drones to tackle a new AI task: object detection. We develop multiple CNNs, trained to recognize two object classes, with different trade-offs between accuracy and throughput. The largest and most accurate model scores a mean average precision (mAP) of 50% on an in-field collected dataset while running at 1.6 frame/s on the nano-drone MCU with a power envelope of only 134 mW. The effectiveness of this model was further validated in-field, where it was integrated with state machine-based exploration policies. These policies, characterized by low computational complexity, leverage data from time-of-flight

distance sensors to perform obstacle avoidance. In these field tests, our system demonstrated an average detection rate of 90%

Building on the progress made in Chapter 5, where we reduced the size of a CNN for visual-based autonomous navigation, we now merge the AI models developed in Chapter 5 and Chapter 6. This integration allows for the sequential execution of visual-based obstacle avoidance and object detection tasks on autonomous nano-UAVs, running in real-time at 1.6 frames/s. Ultimately, this achievement marks the SoA by presenting the first fully autonomous nano-drone tackling a multi-objective mission, which consists of exploring, preventing collision, and detecting objects in real time while relying only on onboard sensory and computational resources. As a result, this thesis brought miniaturized flying robots a step closer to the high-level intelligence of tiny biological systems, demonstrating that it is possible to overcome both the computational and memory burden imposed by ULP MCUs, allowing the simultaneous execution of multiple AI-based perception tasks on nano-UAVs.

Chapter 2

Related works

This chapter outlines the current SoA research on UAVs, categorized by size, weight, power consumption, and onboard processing devices, as detailed in Table 1.1. The latter two characteristics are related to each other, as the budget for onboard electronics is limited to ∼5-15% of the total power envelope of the UAV [29]. Standard-sized UAVs, with a diameter typically exceeding 50 cm and a power budget over 100 W, can afford to host powerful CPUs such as the Intel i7/i5 onboard [30]. Micro-sized UAVs, slightly smaller, have a lower power budget of about 50 W but can still accommodate host powerful embedded devices, such as NVIDIA Jetson/Xavier boards [26, 16, 17]. Both standard- and micro-sized UAVs can afford to execute complex perception and control pipelines onboard, composed of multiple AI tasks running simultaneously [17, 26, 16].

On the other end of the spectrum, nano-sized UAVs have a diameter around 10 cm, weigh only a few tens of grams, and have a power budget of just a few watts [18]. This small platform's size and power constraints prevent hosting high-end computational units. Instead, they have to rely on resource-constrained and simple single-core MCUs. Despite these limitations, the objective of this manuscript is to enable nano-sized UAVs to reach levels of skill and autonomy comparable to those of their larger counterparts. To reach such a level, nano-UAVs must be capable of running multiple AI tasks simultaneously, whereas current SoA only enables executing a single neural network onboard.

The rest of the chapter is organized as follows. Section 2.1 discusses the capabilities of standard- and micro-sized UAVs to execute multiple AI tasks, Section 2.2 examines the current ability of nano-sized UAVs to run single AI tasks, i.e., CNNs, Section 2.3 presents datasets applicable to UAV autonomous navigation, explaining the need for a new one, Section 2.4 reviews algorithms for maximizing the exploration of unknown environments, suitable for simple single-core MCUs, Section 2.5

reviews CNNs for object detection suitable to nano-drones, and, finally, Section 2.6 reviews the approaches for drone racing.

2.1 Standard and micro-sized UAVs

The overwhelming majority of complex robotic perception algorithms have been demonstrated aboard standard- and micro-sized UAVs [30, 31, 12, 32, 16, 33, 34, 26, 25, 17, 35], which feature powerful onboard computers, e.g., GPUs. The sophisticated functionality that can be achieved with these platforms includes onboard autonomous navigation in unstructured natural environments [30, 12] and search for particular objects in the field of view using semantic segmentation, for example, for smart agriculture [31]. More recently, complex visual-based perception pipelines have been demonstrated in the context of drone-racing [16, 33, 34, 26, 25, 17, 35], encompassing the development of CNNs for gate detection, obstacle avoidance, and deep reinforcement learning (RL) approaches for the navigation. These functionalities need multiple CNNs and other non-neural algorithms, such as visual SLAM [32], to enable multiple concurrent tasks, e.g., pose estimation, collision avoidance, and trajectory planning.

2.2 Enabling AI on nano-size UAVs

Nano-sized autonomous UAVs [24, 18, 19], comparable to a hand palm size, are an exciting yet challenging target for the deployment of AI-based autonomous navigation features. Their tiny form factor form (approximately $\sim$10 cm) and lightweight payload (around $\sim$30 g) [1] demand for compact mission computers, such as MCUs, which suffer from severe constraints on both memory (few MB of memory onboard) and computational capabilities. We can distinguish three categories of solutions to enable nano-UAVs with AI capabilities despite these limitations:

1. **Automatic nano-UAVs with offloaded computation**. This approach involves offloading computation to a powerful external base station [40, 41, 42], thereby sacrificing complete autonomy but allowing for more complex processing that the UAV's onboard systems cannot handle;

2. **Simple AI workloads with autonomous nano-UAVs**. This approach aims at fully onboard execution by minimizing the execution workload. Consequently, when using single-core MCUs, the nano UAVs end up achieving only minimal functionalities and/or rely on low-dimensional input signals (e.g., no images) [43, 36, 44, 45, 37, 46, 47];

3. **Accelerated autonomous nano-UAVs**. This approach enhances the onboard computing capabilities with application-specific processors [48, 49, 50,

Table 2.1. Related works based on CNNs, running aboard nano-UAVs. The task is either classification (C), or regression (R).

Work	**Device**	**Description**	**Task**	**Power [mW]**	**Parameters**	**MAC**	**Throughput** [inference/s]
Kooi and Babuška [36]	STM32F405	RL for landing	R	∼250	4.7 k	4.7 k	400
Shi *et al.* [37]	STM32F405	Interaction forces	R	∼250	9.5 k	9.5 k	—
Cereda *et al.* [38]	GWT GAP8	Pose estimation	R	96	303 k	14.7 M	48
Bouwmeester *et al.* [39]	GWT GAP8	Navigation	S	—	171 k	—	5.6
Palossi *et al.* [18]	GWT GAP8	Navigation	C+R	284	320 k	41 M	18
Ours (Sec. 6.1.2)	GWT GAP8	Object detection	C+R	134	4.7 M	534 M	1.6
Ours (Sec. 4.1)	GWT GAP8	Navigation	C+R	102	320 k	41 M	19
Ours (Sec. 5.2)	GWT GAP8	Navigation	C+R	100	2.9 k	1.1 M	139

51, 52] or general-purpose visual navigation engines [24, 19], to enable the fully onboard execution of complex AI workloads.

2.2.1 Automatic nano-UAVs with offloaded computation

To address the computational constraints of single-core microcontrollers (MCUs), one solution involves offloading intensive computing tasks to external, wirelessly connected resources [42, 40]. This method shifts the operational mode of nano-UAVs from autonomous to automatic, as it depends on external infrastructure to complete its tasks. In [42], the authors propose an autonomous navigation approach based on a CNN that uses reinforcement learning to adjust part of its parameters online. The initial values of the weights are obtained by training the whole network with synthetic images obtained from a game engine, and they also prove the in-field functionality using a 80 g drone. However, the action space of their algorithm is limited to the following commands: move 50 m forward, steer 45°, and steer −45°. This results in less smooth and flexible navigation than our approach, which provides a continuous output for the steering angle and adaptive forward velocity. In [40], the authors implement a fuzzy logic position controller and vision-based position estimation by offloading all the computation to an Intel i7 processor streaming images with a 2.4 GHz radio.

This class of approaches, however, suffers from several important drawbacks [14]: *i*) it introduces network-dependent latency, which prevents the drone from operating farther than a few tens of meters from the remote base station, *ii*) the noise on the transmission channel affects the reliability of the transmitted data, *iii*) security becomes a concern for eavesdropping on confidential images and data and denial-of-service attacks on the wireless connections, and *iv*) the power consumption of the high-frequency radio transmission is significant, and the wireless transceiver may dominate the power budget for control (for example, the NINA transceiver of the AI-deck of a Crazyflie nano-drone has a power envelope between 0.3 W and 1 W).

2.2.2 Simple AI workloads with autonomous nano-UAVs

From the second category of solutions, Lambert *et al.* [43] exploit the STM32F4 MCU to implement a simple DL-based *flight-controller* for hovering on a Crazyflie 2.0. Kooi and Babuška [36], a deep reinforcement learning (RL) method employing proximal policy optimization enables autonomous landings of nano-drones on inclined surfaces. The CNN designed for this purpose requires around 4.5 k multiply-accumulate (MAC) operations per forward step. Although the network operates efficiently, computing an inference in approximately 2.5 ms on a single-core Cortex-M4 processor, its functionality remains constrained solely to the landing task.

Similarly utilizing the Cortex-M4 processor, Neural-Swarm [45, 37] leverages a DL-based controller ($\sim$ 27 kMAC) to manage close-proximity interaction forces encountered during formation flights of nano-drones. With a processing cost of $\sim$ 9.5 kMAC, each nano-drone in the swarm processes only the relative position and velocity data of surrounding UAVs, tackling safe maneuvers in close proximity with other UAVs. Zhao *et al.* [44] implement a CNN to improve the nano-UAV's localization accuracy by modeling the localization system's sensor biases. This model can run at 200 Hz on the onboard ARM Cortex M4 MCU requiring about 2.7 kMAC/frame, i.e., 10000$\times$ less operations than existing SoA CNN-based autonomous navigation workloads, e.g., $\sim$40 MMAC/frame in [18].

McGuire *et al.* [46] propose a lightweight navigation algorithm that enables a swarm of drones to explore an indoor area while avoiding collision with the walls by exploiting four laser distance sensors on each drone. The simple sensor input (i.e., four single laser beams) results in an avoidance mechanism that can only detect large and homogeneous obstacles. Daramouskas *et al.* [47] proposes an approach for navigation with obstacle avoidance using a random forest classifier. They report a classification accuracy of 90% while using a size of 229 nodes for the decision tree. However, their approach was only developed and tested with synthetic data generated with the aid of a simulator. While useful for some specific tasks, all of these tiny models are not viable solutions for more challenging navigation problems like the ones we tackle in this thesis.

2.2.3 Accelerated autonomous nano-UAVs

A possible solution to the limitations imposed by single-core MCUs is augmenting nano-UAVs with better compute functionality. For larger UAVs, this is a common choice – using devices such as NVIDIA GPUs, Intel Myriad, or Google Edge TPU [30, 31, 53, 54] that are both flexible and highly efficient. For nano-UAVs, however, the possibilities are more limited. Some recent works emphasize the efficacy of application-specific integrated circuits (ASIC), which are suitable for autonomous navigation functionalities [51, 52, 48, 49, 50] on a low-power budget. Some of these systems have been designed to tackle specific UAV applications, such as visual-inertial odometry (VIO) [48] and simultaneous localization-and-mapping (SLAM) [49, 50], within a power envelope of few hundred mW. While extremely efficient, these systems are inflexible, and they do not implement end-to-end flying functionality but only accelerate some sub-functions, requiring a mission and flight controller MCU anyways.

Conversely, the approach we follow in this manuscript involves the use of multi-core flight controllers designed for artificial intelligence workloads Such an approach

addresses the limitations imposed by higher-complexity DL-based workloads by exploiting fully programmable parallel computing accelerators [55]. Parallel ultra-low-power (PULP) processors use small-scale multi-core clusters with 4-16 cores, with an enhanced RISC-V instruction set architecture (ISA), to exploit intrinsic parallelism of vision workloads, including CNNs. The key advantage of this approach is its flexibility and capability to handle end-to-end flight control tasks. The state-of-the-art COTS MCU for nano-drones is the GAP8 SoC [56], an embodiment of the PULP paradigm with a general-purpose 8-core parallel cluster. This fully programmable MCU was successfully exploited to enable the execution of more sophisticated visual-based AI workloads [19, 38, 18, 24], for tasks such as autonomous navigation and pose estimation using CNNs in the range of 10-100 MMAC/frame. Table 2.2.3 summarizes the main characteristics of GAP8 and the most popular Nvidia GPUs aboard micro-sized drones. GAP8 has up to 8000× less RAM memory and 12141× lower peak throughput than the most powerful Nvidia GPU, i.e., the Jetson AGX Orin.

Table 2.1 gives an overview of AI workloads for autonomous nano-drones. By exploiting the GAP8 SoC, Palossi *et al.* [18, 24] developed the SoA CNN for end-to-end visual-based autonomous navigation on a fully autonomous nano-drone. The CNN, namely PULP-Dronet[18], has a computational cost of ∼41 MMAC per inference and requires 320 kB of memory footprint. The autonomous navigation capabilities of this network are assessed by successfully tackling turns and avoiding collisions with dynamic obstacles appearing along the way. *NanoFlowNet* [39] is a CNN for dense optical flow segmentation that aims at the obstacle avoidance task. The CNN has 171 k parameters and runs on GAP8 at 5.6 frame/s, and it differs from the PUL-Dronet approach as it outputs per-pixel information about the obstacles in the scene. The authors test this CNN in the field on a nano-drone but do not provide statistical results about the robustness of the approach nor the average flight speed that they achieved while avoiding obstacles.

[19, 38] exploit GAP8 to demonstrate a fully autonomous nano-drone performing a human pose estimation task. Their CNN, called PULP-Frontnet, predicts the drone's relative pose with respect to a freely moving human subject, allowing the drone to follow the subject's movement. This prediction aims at maintaining a consistent distance in front of human subjects while following their movements, and it only exploits low-resolution grayscale images captured from a front-facing camera. Their best model in the in-field experiments requires 14.7 MMAC and achieves an inference rate of 48 frame/s while consuming 96 mW. Tackling the same problem, Cereda *et al.* [57] exploited neural architecture search techniques to design a tinier version of the PULP-Frontnet CNN, obtaining a 7.4 MMAC and 65 kB model, while retaining good regression performance when tested in-field. Similarly to what we do in Chapter 5.2, they investigate a network architecture inspired by Mobilenet v2 [58].

While previous works enable the execution of single AI tasks, in this manuscript, we advance the SoA of TinyML on nano-UAVs by enabling the execution of multiple CNNs aboard a nano-drone for the first time. To do so, in Chapter 4.1, we automate the deployment of CNN on the GAP8 SoC, demonstrating it on the PULP-Dronet CNN, after applying a more aggressive fixed-point 8-bit quantization. Moreover, we address two main limitations of the PULP-Dronet CNN. First, we highlight in Section 4.1 how this CNN lacks the ability to guide the nano-UAV around static obstacles. This poses a significant limitation in its practical application within challenging real-world scenarios, where navigating uncontrolled environments becomes critical. This missing capability is attributed to the training dataset of PULP-Dronet, as it comprehends multiple sets of images with disjointed labels for the steering and collision tasks. The second limitation of this work is its non-negligible workload of ∼41 MMAC, limiting its throughput to a maximum of 19 FPS. In Section 5.1, we demonstrate that the PULP-Dronet CNN architecture is over-parametrized for the task it solves. In Section 5.2, we enable autonomous navigation with a network that is 168× smaller than PULP-Dronet, leaving plenty of memory and computing resources that can be exploited to enrich a solid autonomous navigation framework with concurrent capabilities.

In Chapter 6, we deploy another high-level intelligence task on nano-UAVs, but with a computational complexity that is one order of magnitude higher than [38]. We deploy a 4.7 MB CNN to perform object detection of two object classes, and they showcased it in the context of an exploration and search mission. Thanks to the resources we freed by developing our Tiny-PULP-Dronet, in Chapter 6, we enable the concurrent execution of the Tiny-PULP-Dronet CNN with the object detection one.

2.2.4 Automatic deployment tools

Deploying multi-MMAC CNNs on an MCU-class device requires coping with a power envelope of few hundreds of mW, a memory of just a few MB or less, and limited peak performance, demanding for a strict co-optimization of the algorithmic, software, and hardware components [59, 60]. The minimization of a DL model can be performed with *i*) specific topological choices, like using depth-wise convolutions [58, 61] or *ii*) using quantization as a compression technique [62, 63] from *float32* down to *int8* (or less), with a net 4× reduction of model footprint. Quantization can also expose more data parallelism exploitable by packed-SIMD instructions [64], improving the final inference throughput and the energy consumption.

Once the network has been optimized for size, addressing the deployment challenge becomes critical. This involves maximizing the utilization of computing resources through three primary strategies: *i*) parallelizing computations, *ii*) managing the

Table 2.2. Review of the main processing devices aboard UAVs.

Processing device	**Device class**	**UAV size**	**RAM Memory**	**memory vs. GAP8**	**Power**	**Peak Throughput**		**Peak throughput vs . GAP8**
GAP8	MCU	nano	8 MB	1 x	1 – 100 mW	22.7	GOps/s	1 x
Jetson Nano	GPU	micro	4 GB	500 x	5 – 10 W	472	GOps/s	21 x
Jetson TX2	GPU	micro	8 GB	1000 x	7.5 – 15 W	1.33	TOps/s	59 x
Jetson TX2i	GPU	micro	8 GB	1000 x	10 – 20 W	1.33	TOps/s	59 x
Jetson Xavier	GPU	micro	16 GB	2000 x	10 – 20 W	21	TOps/s	925 x
Jetson AGX Xavier	GPU	micro	64 GB	8000 x	10 – 30 W	32	TOps/s	1410 x
Jetson Orin Nano	GPU	micro	8 GB	1000 x	7 – 15 W	40	TOps/s	1776 x
Jetson Orin	GPU	micro	16 GB	2000 x	10 – 25 W	100	TOps/s	4415 x
Jetson AGX Orin	GPU	micro	64 GB	8000 x	15 – 60 W	275	TOps/s	12141 x

memory hierarchy through topology-dependent tiling, and *iii*) minimizing data transfer overheads. This step is particularly crucial for MCU devices, where processing units are inherently limited [59]." General-purpose tools such as TFLite for MCUs and Larq [65, 66, 67], as well as vendor-locked tools like STM32 X-CUBE-AI[1] have been proposed to ease deployment on MCUs. For PULP platforms, on which we focus in this manuscript, two deployment tools have been recently introduced: GWT's *AutoTiler*[2], which is partially closed-source, and DORY [59] with PULP-NN backend [68], an alternative open-source academic framework.

In Chapter 4.1, we exploit these recent advancements to bring the deployment of DL-based visual navigation on nano-drones from handcrafted and hand-tuned deployment [18] to a new streamlined, automated methodology. We leverage DNN deployment frameworks by integrating them into our flow, and we achieve significantly improved performance and energy efficiency on autonomous navigation DL workloads, improving the nano-UAV in-field behavior and freeing resources for even more complex missions and tasks.

2.3 Datasets for nano-UAV autonomous navigation

In Section 5.2 we highlight the importance of introducing a new dataset for visual-based autonomous nano-drone navigation. Therefore, in this section, we review the dataset for UAVs available in the literature. Dupeyroux *et al.* [69] released a dataset for obstacle detection and avoidance, specifically targeting micro-sized drones (i.e., weighting $\sim 0.5\,\text{kg}$). It has 92 GB of data, including high-resolution images (full-HD) for $\sim 80\%$ of the acquisitions. Thus, adapting these images to the nano-drone use case would require a photometric augmentation pipeline to convert full-HD images to the format of a low-quality and low-resolution camera commonly found on nano-UAVs [18]. Moreover, in [69], the obstacle avoidance information is based on radar and distance sensors. Thus, using this information relegates the CNN to solve the autonomous navigation task in a mediated way. Conversely, we collect a dataset logging the low-quality images from a nano-drone along with the pilot's yaw-rate input, which is fed to the flight controller during the data acquisition, allowing us to train an end-to-end network for autonomous navigation.

The Dronet dataset [70] was created to enable autonomous navigation on large UAVs. They combined two sets of images: one from driving cars, where each image was associated with the car's steering wheel angle, and a second one from bicycles,

[1] `https://www.st.com/en/embedded-software/x-cube-ai.html`

[2] `https://greenwaves-technologies.com/manuals/BUILD/AUTOTILER/html/index.html`

where the labels were labeled with the binary collision information. This dataset was successfully exploited to enable autonomous nano-drone navigation in [18], where additional $\sim$ 1300 grayscale low-quality images collected from a nano-UAV were used to fine-tune the network to the low-quality images provided by the nano-drone's camera. However, this dataset does not provide joint steering and collision labels for the navigation, leading to poor performance when tackling static obstacle avoidance [1], as detailed in Section 2.2.3. Similarly, [71] collected $\sim$15 k images for solving the same task as Dronet. However, this dataset is also divided into two sets of images for steering and collision labels. [72] used the Dronet dataset to train an off-board CNN for nano-drones autonomous navigation, still not tackling the static obstacle avoidance scenario.

The dataset presented in Section 5.2 overcomes these limitations as all the $\sim$66 k collected images feature both collision and steering information, i.e., it does not have disjointed labels. Specifically, we log the input of a human pilot navigating a nano-drone in multiple environments. These labels can be used to train an end-to-end CNN that imitates the behavior of a human pilot. Furthermore, we log the drone's estimated state and label all the images with a binary label for obstacle avoidance. We open-source our dataset, dataset collector tool, and pre-trained models to foster research on autonomous nano-drone navigation.

2.4 Autonomous exploration

In Chapter 6, we develop and compare multiple exploration algorithms on a nano-drone. Consequently, this section reviews autonomous exploration algorithms that are specifically suitable for deployment on nano-drones, considering the constraints imposed by the limited onboard resources.

Computer vision-based navigation techniques, based on complex feature extraction pipelines [73] or simultaneous localization and mapping (SLAM) techniques [32], are reliable for autonomous robotic navigation. Still, as SLAM-based approaches have large memory requirements and rely on computationally intensive algorithms, they are exclusive to large UAV systems carrying heavy and high-power embedded computing systems [32].

The state-of-the-art autonomous exploration approaches for nano-drones, which suffer from limited computational power and memory, take advantage of bio-inspired (or bug-inspired) algorithms, which rely on lightweight state-machine-based algorithms and low-power sensor readings[74]. For example, [75] compares two bio-inspired exploration policies: the first changes the heading direction randomly after detecting an obstacle, while the second follows the obstacle's boundaries. Similar to the second one, [74] implements a policy that follows the walls of a room, called wall-following, while [76] describes a spiral motion. Moreover, there is a subcategory

of bug-inspired algorithms that uses ranging measurements for navigating towards a target point [77, 78], even in the presence of obstacles. In Chapter 6, we adapt such bio-inspired ranging-based algorithms to our flying nano robotic platform and, for the first time, we integrate them with a visual object detection pipeline, evaluating the effectiveness of the exploration policy within a search mission.

2.5 Object detection on constrained MCUs.

In Chapter 6, we develop a CNN for visual object detection running fully onboard a nano-drone. Therefore, this section reviews lightweight object detection algorithms that can run on an MCU. Among the CNN-based visual object detection pipelines, single-shot detector (SSD) [79] is a popular approach, consisting of a feature extraction backbone and multiple convolutional detection heads. To reduce the computation and memory costs, shallow backbones have been proposed, e.g., Mobilenet, while preserving the detection scores on widely used dataset [58].

Focusing on the porting of object detectors on embedded systems, Tran Quang Khoi *et al.* [80] deploy an SSD network (SSDLite-MobileNetV2) with a similar architecture as the one used in this Chapter onto a RaspberryPi B3+ mounted on a standard-sized drone. The model execution reaches 0.71 FPS, lower than our obtained throughput. More importantly, this solution exceeds both the power (up to 3 W) and size constraints of the nano-drone system considered in this Chapter. Lamberti *et al.* [81] presented an SSD algorithm for license plates detection on a static multi-core MCU node (GAP8), achieving up to 1.6 FPS with a power consumption of 117 mW. Starting from this previous work, we design and integrate an SSD-based object detection CNN onto a nano-drone capable of exploring the environment while detecting multiple instances and classes of objects in its field of view.

2.6 Drone racing

In Section 4.2, we describe our nano-drone system winning the "Nanocopter AI Challenge" at the *International Micro Air Vehicles* conference 2022, a nano-drone race that focuses on obstacle avoidance (OA) and gate-based navigation (GN). In this section, we focus on these two complex tasks surveying the SoA for various class sizes of drones (Tab. 2.6).

2.6.1 Obstacle avoidance

While racing micro-drones can carry bulky sensors (e.g., Lidars [16], stereo cameras [16, 25]) and GPUs with a power envelope of up to 30 W (Tab. 2.6), nano-UAVs suffer from limited perception capabilities due to their tiny low-power sensors and

Table 2.3. Literature review for obstacle avoidance (OA) and gate-based navigation (GN) on UAVs.

Work	**Size**	**Onboard**	**Task**	**Perception**	**Algorithm**	**Data**	**Compute device**	**Power**	**Competition**
Wagter *et al.* [26]	Micro	✓	GN	Camera	CNN	Real	Jetson AGX Xavier	30 W	AlphaPilot'21
Foehn *et al.* [25]	Micro	✓	GN	Stereo camera	CNN	Real	Jetson AGX Xavier	30 W	AlphaPilot'21
Kaufmann *et al.* [33]	Micro	✓	GN	Camera	CNN	Sim	Intel UpBoard	13 W	IROS'18
Jung *et al.* [16]	Micro	✓	GN/OA	Stereo camera	CV	None	Jetson TK1s	15 W	IROS'16
Pham *et al.* [34]	Micro	✓	GN	Camera	CNN	Sim	Jetson TX2	15 W	—
Palossi *et al.* [18]	Nano	✓	OA	Camera	CNN	Real	GWT GAP8	100 mW	—
Ours (Sec. 4.2)	**Nano**	✓	**OA**	**Camera**	**CNN**	**Sim**	**GWT GAP8**	100 mW	**IMAV'22**

microcontroller units (MCUs) [1]. SoA perception algorithms, such as simultaneous localization and mapping (SLAM) [32], can not run onboard nano-drones due to their steep performance requirements. Even when run off-board, they suffer from substantial performance degradation as nano-drones employ low-quality sensors [82, 83].

Lightweight approaches better suited to nano-drones employ different sensors such as Time-of-Flight (ToF) ranging sensors [46, 6]. These approaches can be implemented onboard and provide robust obstacle avoidance, even in unknown environments, with raw sensor readings or minimal onboard processing (e.g., 30 OP/s in [6]), such as simple bug-inspired lightweight state machines [46]. However, in our competition, only a low-resolution monochrome monocular camera was allowed, narrowing the teams' effort only to visual-based approaches.

A lightweight vision-based system for pocket-sized drones was presented by McGuire *et al.* [84], implementing depth estimation with a stereo camera and achieving obstacle avoidance at low speed (0.3 m/s). The SoA visual-based CNN for autonomous nano-drone navigation is PULP-Dronet [18], trained on real-world data to predict a steering angle and a collision probability based on a QVGA monocular image. PULP-Dronet runs onboard the GAP8 SoC at 19 frame/s, proving in-field obstacle avoidance capabilities up to a speed of 1.65 m/s when coping with a dynamic obstacle. As a result, at the IMAV competition, 3 of 6 teams, including us, employed this CNN as a starting point to build their visual obstacle avoidance pipeline. in Section 4.2, we explain how we modified the PULP-Dronet reference implementation in its task, i.e., the original CNN predicts a collision probability and a steering angle. Instead, our model is optimized to predict three collision probabilities by horizontally splitting the input image into three regions. Then, we introduce a novel training pipeline that exclusively relies on simulation, while the original work uses real-world images based on autonomous driving cars. Finally, we enrich our simulator with a photometric augmentation pipeline, which increased the generalization capabilities of our model to a top-scoring in-field performance. Compared to PULP-Dronet, our approach targets more aggressive obstacle avoidance capabilities: up to 2 m/s speed, concurrent static and dynamic obstacles, and 5 min uninterrupted flight. A thorough discussion on the comparison with the SoA PULP-Dronet is presented in Sec. 4.2.4.

2.6.2 Gate-based navigation

SoA approaches for micro-drone racing competitions tackle trajectory planning and optimization, gate detection, and control [35]. However, their prohibitive complexity prevents them from being implemented on nano-drones, even without considering the computational budget needed by the obstacle avoidance task. Time-optimal

trajectory optimization relies on model predictive control (MPC) solving linear algebra at high-frequency (∼100 Hz) and with real-time constraints [85]. To cope with the MPC complexity, Foehn *et al.* [85] exploit an NVIDIA Jetson TX2 GPU, which has ∼ 60× higher computation capabilities than GAP8.

For gate detection, the SoA exploits either *i*) CNNs for segmentation of high-resolution images [25], which results in a computational complexity of more than 3 GOPs [25] per inference, or *ii*) traditional computer vision approaches with stereo images [16]: both are still out of reach for nano-drones. Kaufmann *et al.* [33] proposed a simpler CNN for predicting the gate's poses directly from the image, but this technique runs at only 10 Hz on a powerful (∼13 W) Intel UpBoard and relies on a coarse map of the gate's positions.

PencilNet [34] is a lightweight CNN for gate pose estimation trained on simulated images, addressing the sim-to-real gap with an intermediate image representation. This model comprises 32 k parameters and 53 kMAC operations per frame. From a computational/memory point of view, this CNN is suitable for real-time execution on our nano-drone. However, the gate pose estimation is only part of a more complex pipeline to achieve gate-based navigation, which additionally requires a memory-consuming mapping of the environment and a more complex trajectory planning. For this reason, the PencilNet CNN was demonstrated on a drone equipped with a powerful Nvidia Jetson TX2, with a power consumption 75× higher than our nano-drone's GAP8 SoC, and a high-end Intel RealSense T265 for the state estimation.

More lightweight approaches for visual servoing have also been demonstrated aboard autonomous nano-drones, based on simple computer vision approaches, such as color segmentation, to cope with the platform's limited computational resources. In [86], the authors used raw image data to detect and fly through monochromatic gates, while in [87], a simple target-tracking algorithm for monochromatic objects was introduced. However, neither work targets high-speed scenarios, resulting in too limited agility for drone racing.

Chapter 3

Background

This Chapter goes through the nano-drone platform used throughout the whole thesis, i.e., the Crazyflie 2.1 by Bitcraze, and the processor used to enable the execution of multiple AI tasks aboard, i.e., the GAP8 SoC.

3.1 Nano-UAV robotic platform

The robotic platform employed in this manuscript is the commercial-off-the-shelf (COTS) Crazyflie 2.1 nano-quadrotor from Bitcraze[1]. Figure 3.1 gives an overview of the system. This tiny UAV weighs 27 g, has a diameter of ∼10 cm, and a total payload of ∼ 15 g. This open-source and open-hardware drone uses the STM32F405 MCU as its flight controller coupled with the Bosch BMI088 inertial measurement unit (IMU), which combines an accelerometer and gyroscope. The STM32 MCU operates at speeds up to 168 MHz and integrates 48 kB SRAM and 128 kB flash, and it is in charge of all low-level flight controller functionalities, such as sensors' interfacing, state estimation, and low-level control. The IMU data drives an extended Kalman filter for state estimation at a rate of 100 Hz, while a proportional-integral-derivative control loop cascade manages actuation. This cascade comprises two control loops, with one governing attitude at 500 Hz and the second updating position at 100 Hz. The Crazyflie 2.1 also integrates a nRF51822 MCU for 2.4 GHz ISM band radio communication. As our drone is completely autonomous while performing its mission, we only use this radio communication for the purpose of dataset collection in Section 5.4.2.

Our configuration extends the robotics platform with two COTS pluggable printed

[1]https://www.bitcraze.io/products/crazyflie-2-1

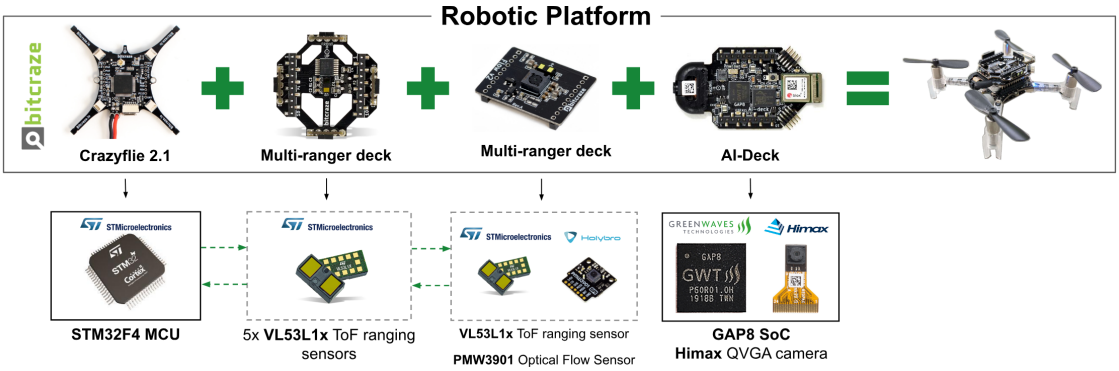


Figure 3.1. The robotic platform comprises the Crazyflie 2.1, Flow deck, Multi-ranger deck, and AI-deck.

circuit boards (PCBs) from Bitcraze: the Flow deck and AI-deck[2]. The 3.5 g optical flow sensor features a low-resolution, down-looking PMW3901 optical flow visual sensor, enabling the drone to detect motions in the 2D plane parallel to the floor, coupled with a VL53L1x time-of-flight (ToF) ranging module, which provides a distance measurement from the ground. These inputs enhance onboard state estimation accuracy, minimizing long-term drift.

The second expansion board, the AI-deck, is the commercially supported version of the PULP-Shield research prototype, introduced in [24]. This PCB, weighing 4.4 g, is the primary onboard computing unit responsible for executing heavy AI workloads, such as the CNNs proposed in this manuscript. The PCB extends the nano-drone's onboard capabilities with an energy-efficient GAP8 processor [56], off-chip DRAM, and Flash memory (8 MB and 64 MB, respectively), a QVGA resolution low-power gray-scale camera (i.e., Himax HM01B0 sensor), a UART communication channel between the STM32 and the GAP8, and a versatile ESP32-based WiFi module. In this thesis, we focus on a configuration where the power-intensive Wi-Fi remains off, aiming for a fully autonomous system where all navigation intelligence resides onboard the nano-drone without relying on external communication or computation. However, we exploited the high bandwidth of the Wi-Fi communication for the dataset collection in Section 5.4.2.

In Chapter 6 and Section 5.2 we extend our robotic platform with an additional PCB, i.e., the *Multi-ranger deck*. This deck weights 2.3 g and comprises five single-beam VL53L1x ToF distance sensors, placed on both the top and lateral sides of the drone, and provide line-of-sight distance measurements within the [0, 4]m range, and they operate at a frequency of 20 Hz. In Chapter 6, we use this deck to feed

[2]https://store.bitcraze.io/products/ai-deck

the exploration state machine with ToF data, performing autonomous exploration and obstacle avoidance, while in Section 5.2 we use this deck for dataset collection purposes.

3.1.1 GAP8 System-on-Chip

The GAP8 SoC [56], which is used for all onboard vision and tinyML processing, is a commercial embodiment of the PULP platform [64]. GAP8 has 1+8 general-purpose RISC-V-based multicore MCU, where the nine cores are organized in two power and frequency domains, namely the fabric controller (FC) and the cluster (CL), as shown in Figure 3.2. The former features one single-core low-power processor implementing the RISC-V `RV32IMCXpulpv2` ISA based on the RI5CY design[64], paired with on-chip SRAM memory and peripherals supporting protocols like SPI, I2C, HyperBus, and Camera Parallel Interface (CPI). These can be accessed through a dedicated Direct Memory Access (DMA) engine called microDMA to offload the communication burden from the FC. Ultimately, the FC is used for control-oriented tasks, acting as an "activity supervisor" managing the interfaces to off-chip sensors/memories and orchestrating on-chip memory operations.

On the other hand, the CL is a fully programmable parallel accelerator designed to execute computationally intensive parallel workloads, such as vision-based CNNs, enabling high-level energy efficiency via the parallel computational paradigm [55]. The CL includes 8 RISC-V cores with the same ISA extensions as the FC; in particular, the `Xpulpv2` extension includes 8-bit and 16-bit packed SIMD, Multiply-And-Accumulate, and dot-product operations, which enhance the SoC's linear algebra capabilities for such workloads. The cores are connected over a logarithmic interconnect to 64 kB of Tightly Coupled Data Memory (TCDM) comprising 16 memory banks. The logarithmic interconnect assures 1-cycle latency access for all the cores when there is no bank conflict, enabling fast data parallelism among the cores. The logarithmic interconnect assures 1-cycle latency access for all the cores when there is no bank conflict, enabling fast data parallelism among the cores. The cluster has a DMA to offload data transfers between the TCDM and the 512 kB L2 memory. The cores are programmed with the Single-Program Multiple-Data programming model and synchronized using dedicated hardware for low-latency barriers. Ultimately, the CL can yield up to 5.4 GOps/s [68] in a power envelope of ∼100 mW.

The FC and cluster domains are separately clocked to tune for best energy efficiency and performance trade-off. The FC domain can be clocked between 50 MHz and 250 MHz, while the cluster domain can be between 100 MHz and 175 MHz. The SoC voltage can be set between 1 V and 1.2 V depending on the FC and cluster clock frequency.

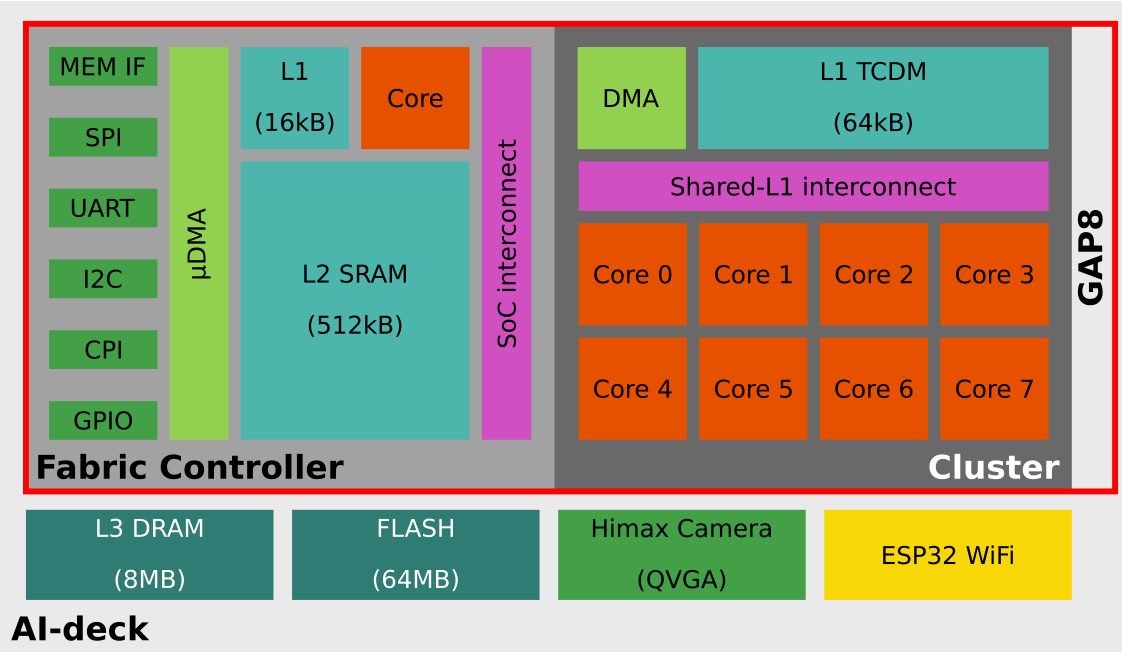


Figure 3.2. AI-deck diagram and the GAP8 System-on-Chip architecture.

Combining the STM32 MCU with the GAP8 SoC enables the heterogeneous architectural paradigm at the ultra-low-power scale [55], enabling the onboard execution of sophisticated vision-based algorithms. In this *host-accelerator* context, the STM32 represents the *host*, handling control-oriented tasks (i.e., flight controller), while the GAP8 offers general-purpose parallel computation capabilities, acting as the *accelerator* for compute-intensive perception and navigation tasks.

Chapter 4

Automating the deployment of AI on MCUs

The challenge of deploying DNNs on nano-UAVs is significantly compounded by their limited computational and memory resources. These constraints necessitate a focus on minimizing and optimizing vision-based DNNs, which traditionally involve error-prone and labor-intensive development processes. Section 4.1 explores methodologies and software tools designed to streamline and automate the deployment of vision-based CNN navigation on an ultra-low-power (ULP) multi-core system-on-chip, serving as a mission computer for nano-UAVs. We exemplify our approach with the deployment of PULP-Dronet [18], a SoA CNN for the autonomous navigation of nano-UAVs. Our pipeline spans from initial training to final closed-loop evaluation, achieving comparable prediction accuracy to the original CNN [18] while significantly enhancing the in-field performance of our nano-drone. Notably, we have managed to halve the memory footprint and increase inference throughput by 1.6 $\times$ by employing an 8-bit quantization scheme instead of the 16-bit original one.

Additionally, Section 4.2 demonstrates the robustness of our automated pipeline for deploying DNNs on MCUs, particularly within the challenging setting of a real-world drone racing competition. Our fully onboard deep learning solution, tackling visual navigation and obstacle avoidance, led to victory at the "IMAV 2022 Nanocopter AI Challenge," the first international competition for autonomous nano-drones. We achieved this result by exploiting the optimization pipeline that will be outlined in Section 4.1, which adeptly addresses the stringent memory, computational, and sensor limitations inherent to the nano-drone platform used in the competition. Additionally, we take an additional step to ease the development of AI capabilities on nano-drones. We mitigate the lack of a dedicated dataset in

the nano-UAV setting, which stifles the development of novel AI algorithms in this niche, by designing a new CNN that is trained exclusively with simulated data, and we introduce a sim-to-real mitigation strategy for real-world deployment.

4.1 Automated End-to-End Optimization and Deployment of DNNs on nano-drones

In the past years, unmanned aerial vehicles (UAVs) have been adopted in a wide range of applications, such as surveillance and inspection of hazardous areas [18, 88]. Nano-size UAVs, with a form factor of a few centimeters and a weight of tens of grams, are the ideal candidates for fully autonomous indoor navigation as they can safely operate near humans and reach narrow spots with their reduced dimensions [18, 24, 19]. However, these platforms have a total power envelope of a few Watts, of which only $5-15\%$ is allotted for computation, making it challenging to deploy real-time navigation pipelines directly onboard [29]. Furthermore, the small physical footprint and limited payload that nano-UAVs can carry constrain the battery and printed circuit board sizes. Overall, these constraints mean that onboard computing devices must have the physical footprint, power envelope, and on-chip memory of a typical microcontroller unit (MCU).

For traditional UAVs, the classical approach for autonomous navigation is simultaneous localization and mapping (SLAM), which creates a map of the environment and plans the trajectory according to it [89]. Classical SLAM is too computationally intensive to be feasible on nano-UAVs. An alternative emerging approach is to infer relevant navigation information directly from onboard sensors and cameras using machine learning-based algorithms. In particular, deep convolutional neural networks (CNNs) have recently proved to provide good performance in autonomous navigation at a fraction of the cost of SLAM: enough to run practical navigation tasks directly on highly resource-constrained platforms [24, 44]. Still, achieving more sophisticated navigation skills requires deploying more complex CNNs under even stricter real-time constraints, promptly reacting to challenging dynamic environments, avoiding collisions, planning new routes, etc. Therefore, it is imperative to look for strategies to minimize the models' complexity and footprint while maintaining high accuracy.

Recently, low-power multi-core System-on-Chips (SoCs) have been introduced as potentially ideal devices to combine an MCU's flexibility with AI-oriented compute acceleration capabilities [56, 90]. At their peak performance, these devices deliver up to 10–100× better performance and efficiency than conventional MCUs, constituting an ideal platform for fully onboard DNN-driven autonomous navigation. However, their complex architecture, together with the non-trivial requirements of

DNN-based algorithms, requires a complex procedure including training, quantization, and a difficult hand-tuning phase to maximize performance on the final target – a critical step to achieve a high frame rate and thus good in-field navigation performance.

In this chapter, we focus on automating the end-to-end deployment of a DNN-based neural flight controller on top of a nano-UAV employing the GreenWaves Technologies (GWT) GAP8 SoC [56] – one of the most advanced commercially available AI-oriented SoCs suitable to nano-size drones. We evaluate two distinct toolsets available for GAP8, namely the GAP*flow* provided by GWT and the open-source NEMO/DORY flow fostered by the research community [59]. Specifically, we adapt and tune these flows to automatically deploy a CNN for autonomous navigation based on the state-of-the-art (SoA) PULP-Dronet [18]. PULP-Dronet is a residual network used to drive a nano-UAV through an interior (e.g., a corridor) or exterior (e.g., street) environment, deriving a *probability of collision*, used for obstacle avoidance, and a *steering angle* to keep within a lane – implemented as a classification and a regression task, respectively.

Differently from the seminal PULP-Dronet, which relied on 16-bit fixed-point data representation, we focus on fully automated deployment, including network quantization to 8-bits, data tiling, code generation for the GAP8 SoC, evaluation of performance on the regression and classification tasks. We also improve the integration of the new PULP-Dronet with the flight controller, with a more robust approach to deal with situations where the network's output is not providing strong guidance. We compare our results in terms of accuracy to the original PULP-Dronet, showing that the prediction capability is maintained ($\sim$90% for the classification despite the stronger quantization). Our results show a throughput up to 19 frame/s, improved by a factor of up to 1.6$\times$ and total energy consumption of $\sim 3{-}4$ mJ/frame, which is $\sim 44 - 58\%$ less than our baseline.

Moreover, we contribute a thorough exploration of the real-world performance of the CNN in exterior and interior environments, evaluating the drone's adherence to expected behavior in several controlled experiments performed in a room equipped with a Vicon motion capture system. We individually assess the obstacle avoidance and steering capabilities. We find that the drone can stop 0.42 m away from a dynamic obstacle that appears 1.5 m in front of the drone while flying with 1.41 m/s, with a significant 25.3% improvement in the speed/braking-distance ratio vs. the original PULP-Dronet. Furthermore, we also demonstrate the capability of the drone to fly an angled narrow tunnel, and we record the trajectories for various drone velocities. We also evaluate the free-flight capabilities of the drone in a controlled indoor environment, achieving 110 m path in 56 s, which marks an improvement of $\sim 4\times$ vs. our baseline.

Our new, streamlined approach significantly improves the autonomous flight capabilities of PULP-Dronet while freeing up resources (i.e., reduced memory footprint and inference time), which allows the system to handle even more tasks (e.g., localization, detection, tracking, etc.). Also, we investigate the generalization capabilities of the drone flying in new environments that are not captured by the training dataset. Among all considered environments, we recorded the longest flight time of 171 s in an urban street.

4.1.1 Background

PULP-Dronet CNN

Dronet [70] is a vision-based end-to-end autonomous drone navigation CNN, deployed on a nano-drone, for the first time, in the seminal PULP-Dronet project [18]. This shallow NN is based on three consecutive ResNet [91] blocks that branch the last layer to produce two outputs: a probability of collision (classification problem) and a steering angle (regression problem). The CNN was originally developed using 16-bit fixed-point arithmetic and a quantization-aware training process. Images used in the original training/validation/testing, and also in this Chapter, are partitioned into three disjoint sets:

- **Udacity**: ∼39.1K high-resolution images labeled only with steering angle.
- **Bicycle**: ∼32.2K high-resolution images labeled only with collision probabilities.
- **Himax**: ∼1.3K low-resolution images collected from the same camera aboard our target nano-drone and labeled only with collision probabilities.

The union of Udacity and Bicycle sets results in the so-called *Original* dataset that we use to train our PULP-DroNet V2 in PyTorch (100 epochs) and to select the models that minimize both regression and classification error on the validation set.

4.1.2 Deployment Automation Flow

The development of AI-based algorithms on MCU-class processors, aboard a nano-drone, is a complex multi-objective optimization problem that must take into account: *i*) memory availability, *ii*) power envelope, *iii*) hardware limitations (e.g., no FPU), and *iv*) throughput. Therefore, to enable the execution of PULP-Dronet on GAP8 under these constraints, we assemble and streamline a flow of automated tools that divide the process into two main stages: *i*) quantization of the neural network, and *ii*) hardware-aware deployment of the quantized model.

Quantization

This stage, remapping the CNN's numerical representation, e.g., from `float32` to `int8`, enables efficient integer computation on the underlying hardware. From a mathematical viewpoint, the tools we consider in this chapter focus on *uniform affine quantization*: all tensors $\mathbf{t}$ (typically inputs $\mathbf{x}$, outputs $\mathbf{y}$ or weights $\mathbf{w}$) are first restricted to a known range $[\alpha_t, \beta_t)$, then they are mapped to N-bit purely integer tensors $\widehat{\mathbf{t}}$ by means of a bijection:

$$\mathbf{t} = \alpha_{\mathbf{t}} + \varepsilon_{\mathbf{t}} \cdot \widehat{\mathbf{t}}\,, \tag{4.1}$$

where $\varepsilon_{\mathbf{t}} = (\beta_{\mathbf{t}} - \alpha_{\mathbf{t}})/(2^N - 1)$. $\varepsilon_{\mathbf{t}}$ is often called the *scaling factor* used to scale tensors from their floating-point to their integer representation. Quantization flows enforce the representation quantized tensors of all waits and part of the data tensors in the network – the latter typically together with ReLU activation functions.

NNTOOL is the NN mapping flow developed by GWT, included in the GAP*flow*, that converts a TFLite topology graph into a new custom representation. It is distributed as part of the GAP8 software development kit[1]. NNTOOL performs "layer-fusion", post-training calibration and quantization (8/16-bit), and folds batch normalization (BN) into the convolution layer that precedes it, avoiding costly intermediate buffers and saving a small amount of memory traffic (i.e., 1.792 kB in Dronet). On the other hand, NEMO is the quantization tool used by the open-source pipeline NEMO/DORY [59], which provides both post-training quantization (i.e., quantizing the model without further re-training, using only lightweight calibration) and quantization-aware training (i.e., quantization at training-time, to mitigate potential accuracy loss). NEMO does not fold BN layers, but instead, it converts them into fully integer channel-wise scaling operations [60].

For our application, we apply post-training quantization at 8-bit for both NEMO and NNTOOL, which is – to date – the most commonly adopted quantized bit-width and is supported by both flows. Specifically, NNTOOL employs a signed *int8* format for both activations and weights, whereas NEMO employs *uint8* for activations and *int8* for weights. Both tool-sets require a *Conv-BN-ReLU* pattern for all main branches of each ResBlock. This simplifies both quantization and deployment: the accumulated tensor at the output of the Conv operation naturally requires a finer grain representation than that of inputs and weights – both flows employ 32 bits. Integer scaling and ReLU can be applied to a single element at a time, meaning that there is no need to materialize a full tensor of 32 bits elements – rather, each element is produced at 32 bits by Conv but immediately reduced to 8 bits by ReLU or BN+ReLU. The baseline version of the NEMO flow does not support the quantization of data that is not at the output of a ReLU;

[1] `https://github.com/GreenWaves-Technologies/gap_sdk`

as a consequence, we introduce a further modification by pushing the final ReLU of the ResBlocks back to the residual branch. Figure 4.1 summarizes the minor modifications that were used in the two flows with respect to PULP-DronetV1 – establishing two new NN topologies, namely, PULP-DronetV2 GAP*flow* and PULP-DronetV2 NEMO/DORY.

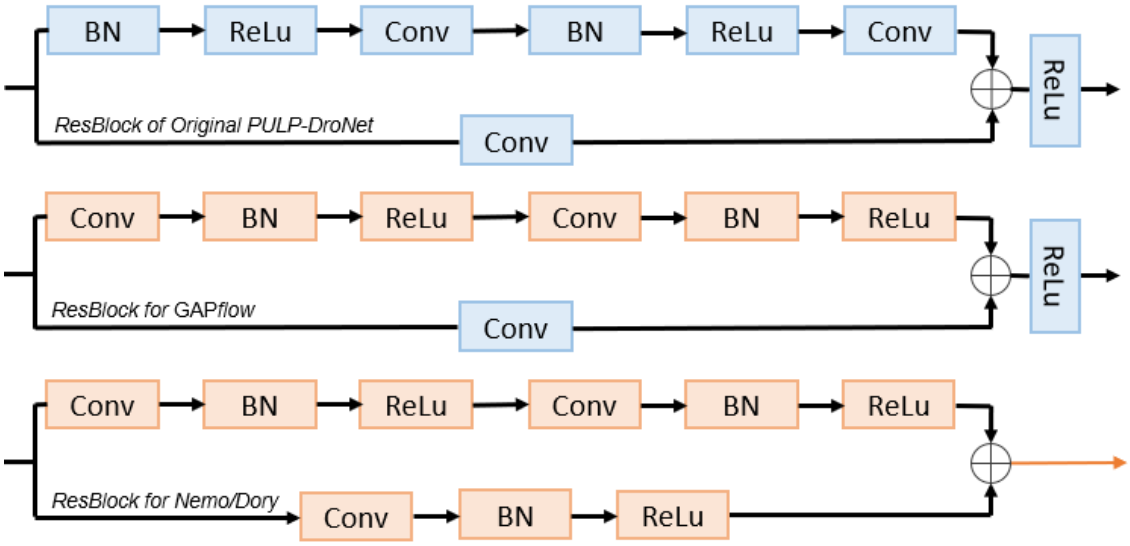


Figure 4.1. ResBlocks of PULP-DronetV1, PULP-DronetV2 GAP*flow* and PULP-DronetV2 NEMO/DORY (top-bottom order).

Hardware-aware deployment

The deployment goal is to enable and exploit the hardware platform by generating C code that: *i*) maximizes the parallel execution over all available cores, and *ii*) minimizes the data transfer overhead. On GAP8, the main challenge is the limited L1 memory (64 kB), which forces the deployment tools to solve an optimization problem, partitioning the tensors into smaller chunks of data, called tiles, to be moved between L2 and the L1 memory.

Both GAP*flow* and NEMO/DORY partition this problem in two separate parts: *i)* a set of optimized kernels operating exclusively on L1 data tiles, and *ii)* a tiling solver to define the optimal size for tiles and generate the code for the related data transfers between L2 and L1, including double buffering for all tensors. As optimized primitives, GAP*flow* relies on a set of open-source kernels available within the GAP SDK, possibly defining custom ones. NEMO/DORY uses the open-source

PULP-NN library[2] [68]. The tiling solver employed by the GAP*flow* is a proprietary tool called AutoTiler, whereas NEMO/DORY employs DORY, an open-source flow [59].

The two primitive libraries exploit different data layouts, affecting the final performance on the Conv layers. PULP-NN, employed by NEMO/DORY, exploits the height-width-channel (HWC) layout, where the data along the channels' dimension is stored with a stride of one, while the data along the width dimension is stored with a stride equal to the number of channels. NNTOOL uses the channel-height-width (CHW) format, reverting the previous order. The convolutional layer can be performed either as a *direct convolution* or as a *matrix-matrix multiplication*, optimized for CHW and HWC layouts. Both implementations have pros and cons. Direct convolution uses a sliding window with a masking/shuffling mechanism, while in the matrix multiplication case, we need to pay some extra overhead to rearrange the input data to a single-dimension tensor (i.e., `im2col`, image-to-column) so that the convolution can be computed as matrix multiplication. On the other hand, matrix multiplication is a more regular operation than convolution, it is essentially identical for any filter size, and it does not require any data shuffling. In general, the HWC layout and the matrix-matrix multiplication become more convenient when the feature map of the convolved layer has many input channels. Conversely, the CHW data layout used by the GAP*flow* is most advantageous with direct convolutions on Conv layers with spatial dimensions much larger than the number of input channels.

The tiling solver employed by GAP*flow* is the AutoTiler, whereas the open-source flow employs DORY [59]. AutoTiler is a proprietary, partially closed-source tool. It can automatically promote full tensors from L3 to L2 and L1 or tile them to maximize performance using the GAP*flow* backend primitives. On the other hand, DORY specifies tiling as two separate problems – one for L3/L2 and the other for L2/L1 transfers. To promote data from L3 to L2, DORY uses a set of simple heuristics, such as looking at the known-good solution first (e.g., copying the full weights for the next layer in L2 while the current one is being run, keeping all activations in L2) and revert to less optimal ones when the former ones are not feasible (e.g., move part of the activations in L3). For the L2/L1 transfers, insisting on a much smaller L1 size (64 KB), tiling is specified as a constrained optimization problem with the objective of maximizing L1 utilization and, at the same time, maximizing a few hardware-aware heuristics (e.g., favor tiles that are better parallelized due to their specific sizes).

Overall, we observe that for our CNN, the AutoTiler finds a better solution for layers that are spatially large and without many input channels, such as the first

[2] `https://github.com/pulp-platform/pulp-nn`

convolutional layer; DORY, on the other hand, performs better for layers where the number of inputs channels is high. By inspection of the generated code, we also notice that the AutoTiler can fuse consecutive layers (e.g., convolutions, max-pooling, and ReLU) and apply multiple operations directly on the same L1 tile, avoiding an intermediate copy to L2. This is the case for the first part of PULP-Dronet (i.e., Conv5 × 5 + MaxPool), where the AutoTiler merges the first two layers, while DORY executes them one after the other, as it can not store in the L1 memory all the needed parameters required by the HWC layout. Both GAP*flow* and NEMO/DORY implement, whenever possible, pipelined memory/computation phases using the GAP8's μDMA (L3-L2).

Platform integration & low-level control

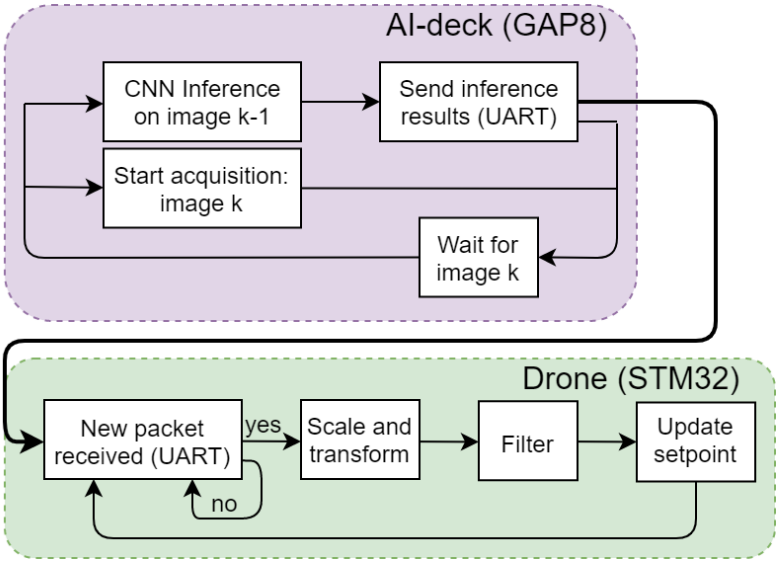


Figure 4.2. Overview of the main acquisition and control loops. The AI-deck is in charge of image acquisition and perception, and the drone's MCU runs the application that interprets the perception results and transforms them into flight commands

.

To enable autonomous navigation on the nano-drone, the inference results of the CNN running in the AI-deck have to be communicated to the flight controller, running on the Crazyflie's main board. This controller runs control algorithms that drive the drone and run on top of the STM32F405 MCU. The Crazyflie flight controller and the AI-deck communicate via UART communication at 115200 baud. Figure 4.2 shows the stages of perception and control. In the AI-deck side (purple), whenever a new inference is started for the current image (k) in the CL, the FC also commands the acquisition of the next image (k+1), which will serve as input

for the next inference. When a new inference result is available, the AI-deck sends this data via UART. When using GAP*flow*, the outputs of the deployed CNN are also quantized at 8 bits: the inference result, therefore, simply consists of 2 bytes - one for the probability of collision and one for the steering rate. When using NEMO/DORY, the accumulated values after the final layer – represented as 32-bit integers – are directly used as outputs. In this case, the inference result is sent as a packet of 8 bytes.

The program that allows receiving the data from the AI-deck and processing it to drive the drone is integrated as a new task in the drone's firmware. To achieve a computational-efficient data exchange, the drone's MCU uses a DMA mechanism to receive the UART data from the AI-deck. The DMA is configured to trigger an interrupt whenever a certain number of bytes has been received – which is two in our case. When the interrupt is triggered, a binary flag (i.e., 0 or 1) that indicates new data available from UART is set. The main loop of the application evaluates the value of this flag every 5 ms, and in case it is set, it reads the two received bytes and then resets the flag back to 0.

The main overview of the inference post-processing stages is given in Figure 4.2, and each step (scale, transform, filter) is detailed by Listing 4.1. First, the two pieces of data ($data[0]$ and $data[1]$) associated with the output of the CNN are de-quantized by multiplying them by the scaling constants resulting from the quantization process. The scaling constants are programmed in the drone's MCU firmware. Next, $I(k)$ is computed, which is an integral term that is added to the probability of collision (p_{col}), and it is meant to penalize the lasting effect of the obstacles in the field view. We noticed that the CNN is sometimes unsure about particular frontal obstacles, and the probability of collision oscillates from values > 0.8 to values below 0.3. Thanks to the addition of the integral term, when the CNN is indicating an obstacle with a $p_{col} > 0.3$, I will increase over time, building up the drone's confidence that it is facing an actual obstacle. When the CNN's inference indicates an obstacle-free horizon ($p_{col} < 0.3$) for a longer time again, $p_{col} - 0.3$ is negative, and therefore, the integral decreases. We clip the value of $I(k)$ to the interval [0,3) as negative confidence is meaningless (on the lower side), and excessive confidence could result in a windup effect. We scale $I(k)$ by a scaling constant w that establishes how much impact the integral has on the final value of the probability of collision. In our experiments, we set this value to 0.2.

To convert the probability of collision p_{col} into forward velocity (v_{unfilt}), we use a simple square low – penalizing velocity quadratically with respect to p_{col}. Furthermore, to reduce the high-frequency noise associated to v_{unfilt}, this value is filtered using a first-order, low-pass infinite impulse response (IIR) filter defined by the coefficient α_1 (we use $\alpha_1 = 0.6$. The same type of filter (defined by α_2) is also used for the steering rate ω_{steer}. We use $\alpha_2 = 0.7$; we observed experimentally that a

lower value results in an increased delay and in low-frequency oscillations around the navigation path. The filtered values for the forward velocity and the steering rate (v_{set} and ω_{set}) represent the new flight setpoint, which is transmitted to the drone's flight controller.

```
while 1:
  if uart_data_available:
      # reset the flag
      uart_data_available ← 0
      # scale data
      p_col ← data[0] * c_scale0
      ω_steer ← data[1] * c_scale1
      # compute the integral
      I(k) ← I(k + 1) + (p_col − 0.3)
      I(k) ← clip_[0,3)(I(k))
      # transform: compute forward velocity
      p_col ← p_col + w · I(k)
      v_unfilt ← v_target(k) · (1 − p_col)^2
      # filter forward velocity
      v_set(k) ← α_1 · v_unfilt + (1 − α_1) · v_set(k − 1)
      # filter steering rate
      ω_set(k) ← α_2 · ω_steer + (1 − α_2) · ω_set(k − 1)
      # command the drone
      command(v_set(k), ω_set(k))
```

Listing 4.1. The listing describes the data processing that is applied to the raw output of the CNN to obtain the setpoint that is communicated to the drone's commander.

4.1.3 Results

In this section, we present three main classes of results: *i*) regression and classification capability of the proposed PULP-Dronet V2 CNNs; *ii*) onboard power analysis and inference performance; *iii*) in-field closed-loop control accuracy and the real-time performance.

Regression & classification performance

In Table 4.1, the NNs quality metrics are reported as accuracy for the classification problem and root-mean-squared error (RMSE) for the regression one, using the Original and Himax datasets for both training and testing. We also evaluate the impact of quantization w.r.t. floating-point calculation for each model proposed and each training set. Particular attention should be given to the scores achieved on the Himax testing set as it maps the type of images available on our flying drone.

Table 4.1. Regression & classification – in **bold** our best fixed8 scores

Training		**Testing**			
NN topology	Dataset	Precision	*Original Dataset*		*Himax Dataset*
			RMSE	Acc	Acc
V1	Original	Float32	0.105	0.945	0.845
		Fixed16	0.097	0.935	0.873
	Original +Himax	Float32	0.109	0.964	0.900
		Fixed16	0.110	0.977	0.891
V2 GAP*flow*	Original	Float32	0.126	0.915	0.831
		Fixed8	**0.124**	0.916	0.840
	Original +Himax	Float32	0.136	0.925	0.881
		Fixed8	0.135	**0.925**	**0.886**
V2 NEMO/ DORY	Original	Float32	0.146	0.902	0.841
		Fixed8	0.143	**0.903**	0.836
	Original +Himax	Float32	0.118	0.893	0.905
		Fixed8	**0.120**	0.892	**0.900**

Testing on Original Dataset

When training on the Original dataset with a `float32` format, both the proposed models show a lower performance w.r.t. the PULP-Dronet V1. The drop is between 0.02 and 0.04 in RMSE and up to $\sim 4\%$ of accuracy. This small difference can be ascribed to *i*) the differences in the NNs topologies and quantization factors (8-bits vs. 16-bits), and *ii*) a weak CNN's convergence. We attribute this weak convergence to the disjoint training datasets for the two problems (i.e., classification and regression); the relatively large unified CNN front-end, resulting in shared weights for two very different tasks up to the very last layer, may also contribute. Our models reveal a different behavior when training on the Original+Himax dataset (`float32`) than the equivalent models trained on the Original set. The GAP*flow* model (similarly to the V1 baseline) slightly improves the classification performance ($\sim +1\%$ accuracy) at the price of a small reduction in the regression capability ($\sim +0.01$ RMSE); instead, the NEMO/DORY model shows the opposite behavior, i.e., accuracy $\sim -1\%$ and RMSE ~ -0.02. The small differences are mainly because the two pipelines use slightly different topologies (Figure 4.1), due to the different approach to batch normalization in the quantized regime.

Our four 8-bit quantized models show a minimal variation in both RMSE and accuracy metrics (within 0.003 and 0.1%, respectively) compared to the respective `float32` version, which is not the case for the V1 baseline as it improves the RMSE of 0.008 and drops 1% of accuracy. The lower variance is the consequence of the different quantization schemes adopted. In fact, in contrast to the quantization-aware

training used in V1, we do not need to retrain the NNs to change the numerical domain due to our post-training quantization. Moreover, post-training quantization does not require the model's training dataset, enabling a faster and straightforward process for producing the quantized model.

Testing on Himax Dataset

When training on the Original dataset with a `float32` format, all the three NN topologies score a similar accuracy of $\sim 83-84\%$. Instead, training on the Original+Himax dataset, the accuracy of all models increases up to $\sim 88-90\%$, proving the beneficial effect of the dataset extension. Finally, the 8-bit quantization of the V2 models does not affect the accuracy, keeping the same maximum (90%) achieved by the 16-bit quantized baseline (V1), which again shows higher variance. Ultimately, the proposed PULP-Dronet V2 models achieve the same accuracy of the V1 baseline on the in-field-collected Himax dataset, despite the reduced datatype (8-bit vs. 16-bit), leading to a highly desirable 2× reduction in the memory footprint. Including the Himax dataset in the training process does not aim at mitigating any quantization effect, but to ensure a better tuning between the CNN model and the onboard camera. In conclusion, regardless of the testing dataset, the 8-bit quantization preserves the CNN's accuracy unaltered w.r.t to the `float32` representation and allows us to deploy and run our model on the target platform successfully[3].

Power consumption & inference performance

We evaluate the execution time and power traces of the proposed models running them on the GAP8 and using a RocketLogger data logger [92] (64 ksps). For these experiments, the SoC's operating points are FC@50 MHz, CL@100 MHz, VDD@1 V, as the most energy-efficient configuration [18], and FC@250 MHz, CL@175 MHz, VDD@1.2 V to push the system at its maximum performance. The GAP*flow* model processes one frame in 1.05 Mcycle, while the NEMO/DORY model needs 11 % fewer cycles. Since our models use almost the same topology of PULP-Dronet V1, they all compute $\sim$ 41 MMAC/frame as the original baseline. We mention, however, that each MAC in PULP-Dronet V2 is an 8x8-bit MAC rather than a 16x16-bit MAC as in V1.

Figure 4.3 shows the power consumption for one frame inference for both models (most energy-efficient configuration or energy-efficient configuration) and highlights

[3]The model's memory footprint could be further reduced with stronger quantization, e.g., 4/2/1-bit; however, this approach is not guaranteed to be sufficient to maintain the full precision regression/classification performance as shown by our work and by the SoA when adopting 8-bit quantization [63, 62].

the time intervals associated with the execution of each CNN layer. The GAP*flow* model (Figure 4.3-A) shows an initial extra stage to normalize the 8-bit input data-range to [-127,+128], as well as some cluster idleness at the beginning of layer 7, due to a μDMA transfer wait. A major difference between the two models is visible in the first two layers (i.e., conv5×5 and max-pool), as the GAP*flow* model achieves better performance merging them (see Section 4.1.2). Nevertheless, NEMO/DORY outperforms its counterpart during the Conv+ReLU pattern, present in each ResNet block.

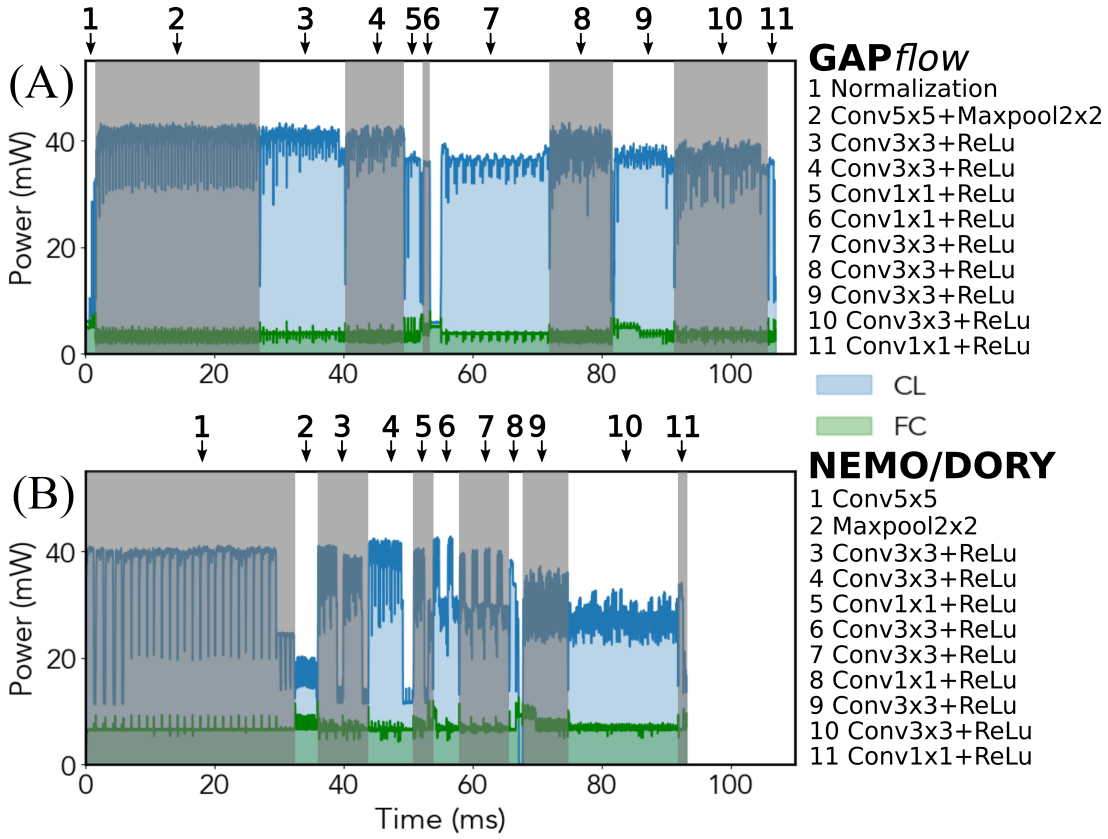


Figure 4.3. GAP8's power waveforms for: FC@50 MHz, CL@100 MHz, VDD@1 V, i.e., the most energy efficient configuration.

In the most energy-efficient configuration, the GAP*flow* and NEMO/DORY models achieve similar performance for one frame inference, as ~ 9 frame/s @ ~ 40 mW and ~ 10 frame/s @ ~ 35 mW, respectively, improving the throughput vs. PULP-Dronet V1 ($40\% - 60\%$). Running the same test, with the SoC's maximum frequencies, the GAP*flow* model scores ~ 17 frame/s @ ~ 119 mW, while the NEMO/-DORY one peaks at ~ 19 frame/s @ ~ 102 mW. Even if the GAP*flow* model exposes a more balanced utilization of the available cores, i.e., almost constant CL power consumption, it pays the overhead for the CWH layout applied to small

WH. Conversely, DORY, even with a less balanced parallel workload, i.e., scattered profile inside layers 3, 4, 5, 6, and 7, reduces overheads due to its HWC layout.

Ultimately, quantization is a key-enabler technique to fully deploy a DNN model on resource-constrained COTS MCUs, which usually lack floating-point units (e.g., the GAP8 SoC). For the same reason, it is hard to precisely compare the execution performances of a full-precision model vs. a quantized one on such a processor. An option is represented by soft-float emulation of all floating-point operations; although, this approach would introduce a major execution overhead. Therefore, we show how quantization improves memory footprint and inference throughput by comparing the proposed 8-bit model to the quantized V1 baseline (16-bit). On the one hand, the 2× reduction in the data-type format halves the total size of parameters from 0.64 MB to 0.32 MB. On the other hand, it allows for efficient exploitation of the GAP8's SIMD instructions, resulting in a throughput speedup of 1.5-1.6× w.r.t. the baseline. This mismatch in speedups (i.e., memory footprint and throughput gain) can be ascribed to multiple factors, such as *i)* non-MAC and non-accelerable operations, and *ii)* non-idealities, e.g., imbalanced workload and marshaling overheads.

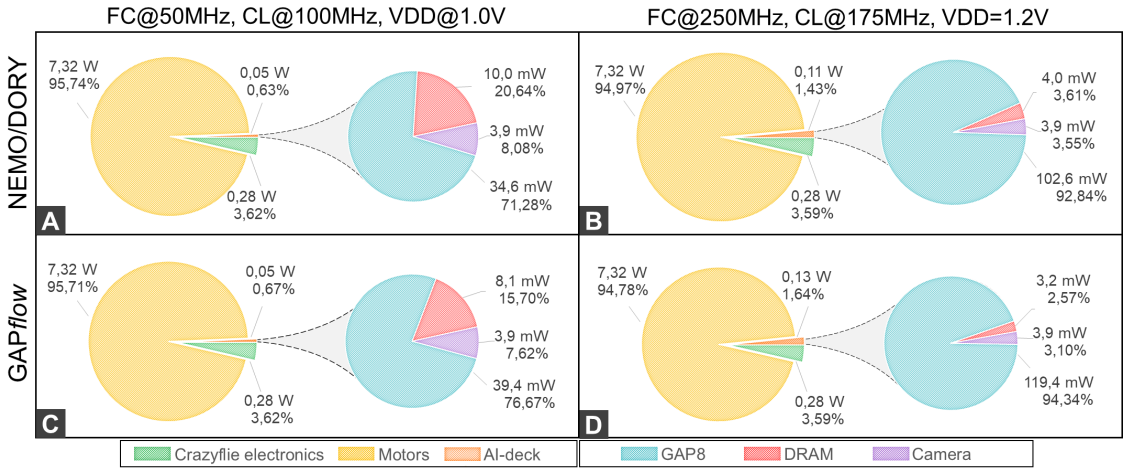


Figure 4.4. The nano-drone's power envelope break-down, with AI-deck zoom-in. A/B) NEMO/DORY, and C/D) GAP*flow* framework. SoC running at FC@50 MHz, CL@100 MHz (A/C) and FC@250 MHz, CL@175 MHz (B/D), the most energy efficient and maximum performance configurations, respectively.

State-of-the-Art comparison

We validate the two proposed GAP8's pipelines, comparing their performance against one of the most popular CNN libraries for MCUs: CMSIS-NN [67]. CMSIS-NN peaks at 0.71 MAC/cycle on 8-bit data convolutions [67] on a CNN's layer similar to our 3×3 convolution, for which we achieve at best 0.81 MAC/cycle/core and

1.0 MAC/cycle/core for the GAP*flow* and NEMO/DORY, respectively. Considering all the inevitable non-idealities, such as sub-optimal load balancing, we yield a weighted performance, for the entire CNN, of 3.9 MAC/cycle (GAP*flow*) and 4.7 MAC/cycle (NEMO/DORY), employing all the GAP8's cores. To concertize this comparison, we consider the CMSIS-NN on top of a high-performance Cortex-M7-based single-core STM32H723VE, which can achieve up to 390 MMAC/s @ 203 mW, running at 550 MHz. The GAP8, at the maximum frequency of CL@175 MHz, achieves 1.44 GMAC/s @ 102 mW and 1.14 GMAC/s @ 119 mW with NEMO/-DORY and GAP*flow*, respectively. This analysis shows that, whether using the GAP*flow* or the NEMO/DORY pipeline, the GAP8 outperforms the Cortex-M7-based MCU with CMSIS-NN by more than 1.7× in power consumption and by more than 3× in throughput. The throughput is of high importance because it has a significant impact on the navigation capabilities, as showed and discussed in Section 4.1.3.

Power break-down

This section analyzes the power-breakdown of the entire nano-UAV running both PULP-Dronet V2 pipelines – i.e., NEMO/DORY and GAP*flow*. We frame this investigation also considering – for each PULP-Dronet V2 version – the two GAP8's operating points introduced in Section 4.1.3, called *most energy efficient* and *maximum performance*, respectively running at FC@50 MHz CL@100 MHz, and FC@250 MHz CL@175 MHz.

The analysis in Figure 4.4 refers to three main parts: *i*) the nano-drone's motors, *ii*) its basic electronics running the stock flight controller, and *iii*) the AI-deck executing our visual-workloads. The electronics slice accounts for both flight controller MCUs (i.e., STM32 and nRF51) and all the basic platform's sensors (e.g., IMU, barometer). Additionally, for all four configurations, we also report a break-down zoom-in on the three main AI-deck's components: *i*) the GAP8 SoC, *ii*) the off-chip DRAM, and *iii*) the ULP camera. The reader should note that the DRAM is considered active at full speed only for the time required to copy the CNN's parameter from L3 to L2 and otherwise turned off. Similarly, Flash memory is not considered in this power break-down evaluation, as it is necessary only for the system initialization (i.e., data movement from Flash to DRAM) and then never again used during the drone's mission.

Figure 4.4 shows how the four motors consume the vast majority of the total power budget, i.e., 7.3 W on average, while the rest of the drone's electronics accounts for 277 mW in all four configurations, as they are always kept at the same operative conditions. Conversely, the power consumption for the AI-deck changes depending on both exploration parameters, but it is never higher than 1.64%, resulting in a motors' power consumption between 94.8% and 96.0% of the total budget.

Focusing on the comparison between the two PULP-Dronet V2 versions, we can identify two main behaviors: *i*) the version developed using the NEMO/DORY framework always shows higher average power consumption for the DRAM and *ii*) the GAP*flow*-based version always has a marginally higher average power consumption for the GAP8 computation. In the most energy efficient configuration, the NEMO/DORY implementation accounts for the 0.63% of the system's power consumption (Figure 4.4-A), while GAP*flow* accounts for the 0.67% (Figure 4.4-C). In fact, as shown in Section 4.1.3, the GAP*flow*-based implementation brings, on average, to a slightly higher power consumption compared to NEMO/DORY one, 39 mW and 34 mW, respectively. Moreover, due to the L2 data pre-loading during the initialization stage, the GAP*flow* version accesses the DRAM fewer times than its counterpart, resulting in a lower DRAM power consumption w.r.t. NEMO/-DORY, i.e., 8 mW and 10 mW, respectively.

Moving to a comparison on the SoC's operating points, the max performance configuration gives a slight advantage to the NEMO/DORY version of PULP-Dronet, which makes the AI-deck consuming 1.43% of the total system's power (Figure 4.4-B), while using GAP*flow* the percentage becomes 1.64% (Figure 4.4-D). This small advantage comes from the fact that the NEMO/DORY version, requiring more L3-L2 data transfer w.r.t. the GAP*flow* version, can benefit more from the increased FC's frequency of the max performance configuration. This difference results in a minimal reduction of the AI-deck's power consumption for the NEMO/DORY-based version, as much as 0.04% and 0.21% compared to the GAP*flow*, for the most energy-efficient configuration and the maximum performance one, respectively. Therefore, from a practical viewpoint, both PULP-Dronet V2 versions perform with a very similar power envelope when deployed on our nano-drone. Ultimately, in all four configurations, we can remark how the addition of the AI workload to our quadrotor only accounts for the smallest portion of the power consumption of the entire system, never higher than 1.64%. Such a small impact on the whole system's power budget demonstrates the capability of running the PULP-Dronet V2 at the highest performance point, only marginally impacting the quadrotor lifetime. This enables the possibility to further extend the onboard intelligence with additional tasks (e.g., tracking, detection, localization), aiming at more complex mission objectives.

In-field closed-loop evaluation

In the following, we perform the in-field evaluation of the navigation capabilities of the PULP-Dronet V2, using the implementation generated by the GAP*flow* framework, and deploying it on a Crazyflie 2.1 nano-drone equipped with an additional AI-deck, as illustrated in Section 4.1.1. We focus on four key aspects to assess the performances of our closed-loop nano-UAV: *i*) the obstacle avoidance task; *ii*) the lane following task; *iii*) the longest flight distance in a familiar environment; *iv*) the

generalization capability, testing the autonomous navigation in never-seen-before environments.

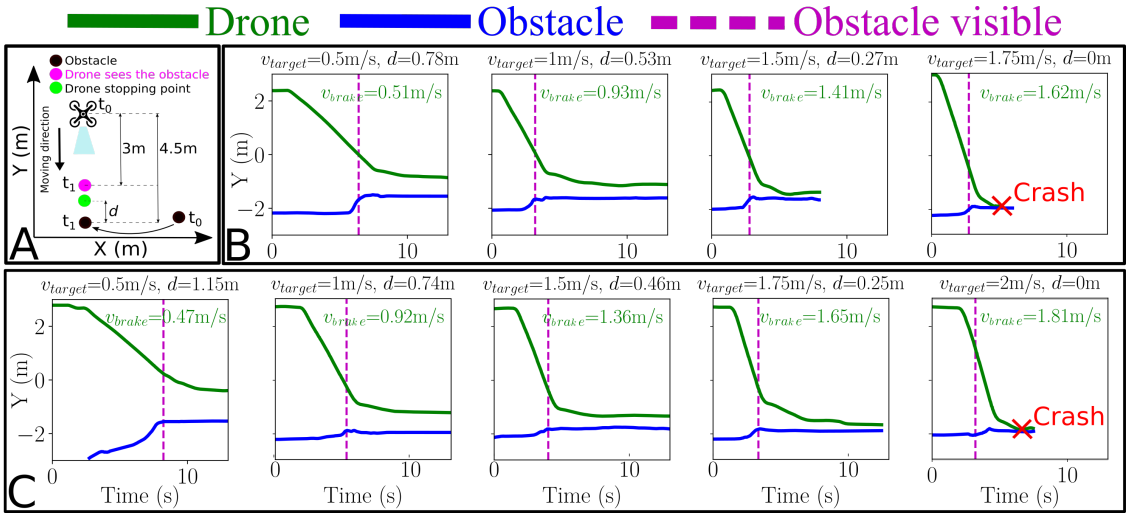


Figure 4.5. A) Experimental setup for the *obstacle avoidance* evaluation. In-field tests are carried sweeping v_{target}, for both the most energy efficient (B) and the maximum performance (C) SoC's configurations.

Obstacle avoidance task

One of the two outputs of the Dronet CNN is the probability of collision used to predict a potential obstacle in the path followed by the nano-drone. In this set of experiments, we assess the drone's robustness in avoiding dynamic obstacles by stressing the closed-loop system with an ad-hoc setup, as shown in Figure 4.5-A. The drone flies a straight trajectory of 4.5 m where a dynamic obstacle (i.e., 0.7×0.7 m cardboard sheet) appears at the end of the path, leaving only 1.5 m for braking and avoiding the collision – i.e., *braking-space*. The straight flight is enforced by silencing the steering angle output of the CNN – i.e., always 0. We perform and record all experiments in a room equipped with a mm-precise motion capture system @ 50 Hz (i.e., Vicon) to analyze the drone's behavior in post-processing.

We investigate this scenario by sweeping two key parameters: *i*) the drone's *target forward velocity* (v_{target}), i.e., a software parameter representing the forward velocity the drone tries to reach if no obstacle is detected, and *ii*) the CNN's inference throughput by means of the two SoC configurations, introduced in Section 4.1.3, named *most energy-efficient* and *max performance*. This evaluation is depicted in Figure 4.5-B for most energy-efficient configuration (FC@50 MHz, CL@100 MHz) peaking at 8.7 frame/s, and in Figure 4.5-C for the maximum performance one (FC@250 MHz, CL@175 MHz) up to 12.8 frame/s.

We stress the system with a growing velocity v_{target} that takes the following values: 0.5 m/s, 1.0 m/s, 1.5 m/s, and 1.75 m/s in Figure 4.5-B, and 0.5 m/s, 1.0 m/s, 1.5 m/s, 1.75 m/s, and 2.0 m/s in Figure 4.5-C. As the v_{target} is a software parameter, we also report, for each test, the actual peak velocity when the drone begins braking (v_{brake}) recorded with the Vicon. We stop this incremental procedure once we reach the limit for which the nano-drone can not prevent the collision anymore. Additionally, in all plots, we highlight a dashed vertical line marking the time when the moving obstacle appeared in the drone's view.

The system proves to be fully working up to $v_{brake} = 1.41$ m/s and $v_{brake} = 1.65$ m/s, in the most energy-efficient configuration and the maximum performance one, respectively. We can notice how the maximum performance configuration provides a similar safety distance ($d =\sim 0.25$ m) w.r.t. its counterpart, despite their different v_{brake}, thanks to the higher inference throughput (i.e., 12.8 frame/s vs. 8.7 frame/s). Lastly, referring to the PULP-Dronet baseline [24]; for the same FC@50 MHz, CL@100 MHz configuration, we score a 25.3% higher v_{brake}/*braking-space* ratio, confirming not only the successful deployment of our PULP-Dronet V2 on the COTS Crazyflie nano-drone but also increased promptness of the system.

Lane detection task

In the following experiments, we assess the PULP-Dronet V2 capability to predict the correct steering angle under two controlled curvature scenarios: a smooth turn of 45°, and a more challenging 90° (i.e., sharp turn). The setup consists of a path 4.5 m long and 1.3 m wide with a left-side turn in the middle, as depicted in Figure 4.6-A/B with the dotted lines indicating the boundaries of the path. We explore two parameters *i*) the target forward velocity (v_{target}) and *ii*) the CNN's throughput, like in the previous obstacle avoidance experiments, utilizing the two SoC's configurations introduced in Section 4.1.3. The software parameter v_{target} is swept with a granularity of 0.25 m/s for the 45° case, and a smaller 0.1 m/s growing-step for the 90° setup, due to its higher complexity.

Starting from $v_{target} = 0.5$ m/s for the 45° setup (Figure 4.6-A) and $v_{target} = 0.3$ m/s for the 90° one (Figure 4.6-B), we keep increasing the v_{target} until the system reaches its limit (i.e., collision). This exploration defines the actual maximum average velocity (v_{avg}) – including the initial acceleration phase – for which the nano-drone can complete this lane detection task, resulting in:

- scenario 45°, configuration FC @ 50 MHz CL @ 100 MHz: maximum $v_{avg} =$ 1.29 m/s;
- scenario 45°, configuration FC @ 250 MHz CL @ 175 MHz: maximum $v_{avg} =$ 1.47 m/s;
- scenario 90°, configuration FC @ 50 MHz CL @ 100 MHz: maximum $v_{avg} =$

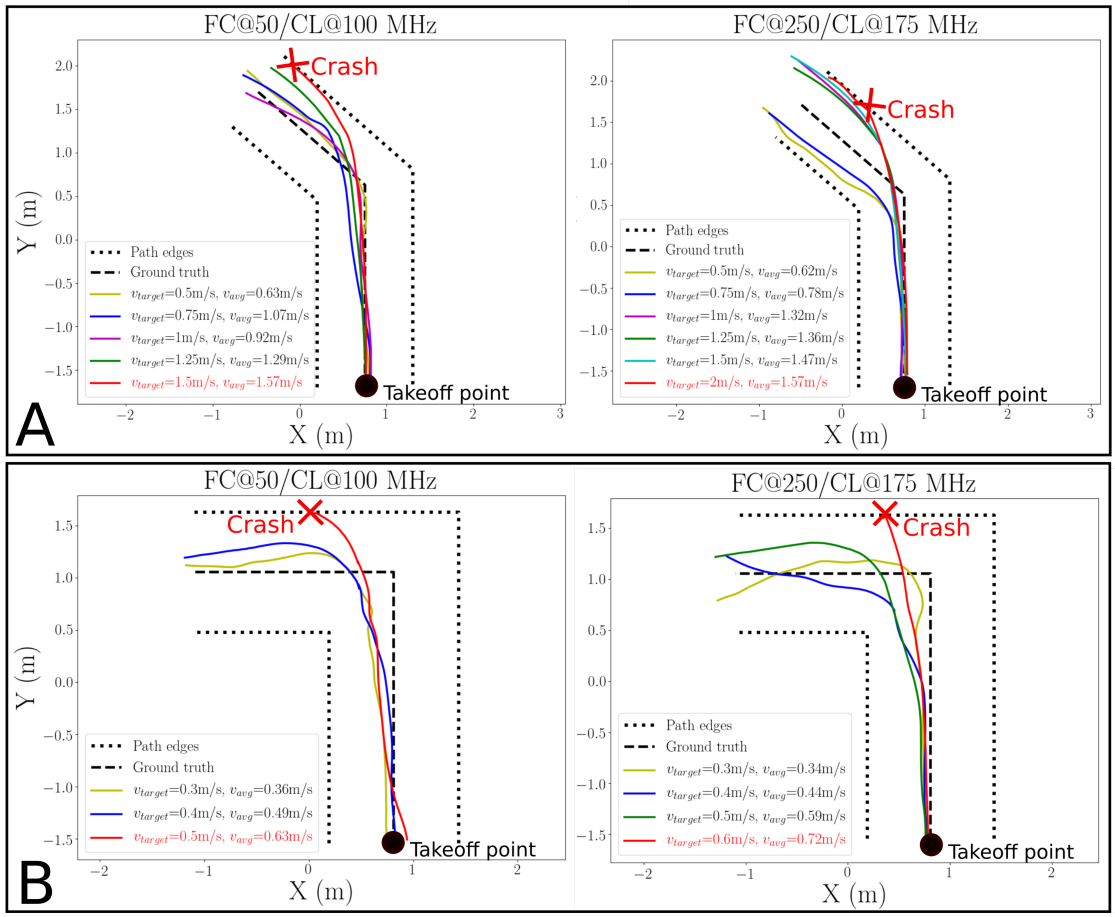


Figure 4.6. Lane detection task evaluation. We assess the CNN's capability of predicting the correct (ground truth) steering angle in a scenario featuring a left–side turn – 45° (A) 90° (B) – at the center of the path. We sweep the forward target velocity (v_{target}) identifying the limit of our system – in red failing configurations.

0.49 m/s;

- scenario 90°, configuration FC @ 250 MHz CL @ 175 MHz: maximum v_{avg} = 0.59 m/s;

The increased throughput of the *maximum performance* configuration (12.8 frame/s vs. 8.7 frame/s), enables higher flight speed in both testing scenarios. As expected, the higher complexity of the 90° scenario is confirmed by a lower maximum v_{avg} compared to the 45° counterpart.

The 2D trajectories of all tests are reported in Figure 4.6, where we define as *ground truth* (dashed lines) the trajectory an ideal nano-drone would follow, keeping the path center for the entire flight. Comparing the actual trajectories with the ground

Table 4.2. Steering angle RMSE and actual average velocity.

		v_{target}[m/s]	SoC configuration			
			FC@50/CL@100 MHz		FC@250/CL@175 MHz	
			RMSE [m]	v_{avg}[m/s]	RMSE [m]	v_{avg}[m/s]
Turn curvature	45°	0.5	**0.07**	0.63	0.26	0.62
		0.75	0.12	1.07	0.21	0.78
		1	**0.07**	0.92	0.22	1.32
		1.25	0.17	**1.29**	**0.20**	1.36
		1.5	collision	1.57	0.23	**1.47**
	90°	0.3	**0.14**	0.36	**0.11**	0.34
		0.4	0.17	**0.49**	0.16	0.44
		0.5	collision	0.63	0.20	**0.59**

truth one, in Table 4.2, we report the root mean squared error (RMSE) for all tests. For both scenarios, we can see a general trend where the higher the actual velocity is (v_{avg}) the more the RMSE grows, ranging between 0.07-0.26 m and 0.11-0.20 m, for the 45° and 90° case, respectively.

A second interesting trend can be seen in the variation of the RMSE between the two SoC configurations of the 45° scenario. At a first look, it seems that a higher throughput penalizes the system's capability, increasing the RMSE. However, by looking at the drone's trajectories in Figure 4.6-A (right plot), it is clear how the successful tests are clustered into two groups:

- tests v_{target} 0.5 and 0.75 (yellow and blue curves) tend to follow a shorter path trajectory;
- all the other curves exhibit a right-hand drive policy fostered by the dataset labeled with steering angles (i.e., Udacity samples are collected in the US).

In both cases, the ultimate trajectory is slightly away from the ideal central ground truth, increasing the RMSE but still accomplishing the mission.

Longest flight distance

In this set of experiments, we want to assess our closed-loop system's autonomous navigation capability in a free-flight mission, exploring a "friendly" environment. For this purpose, we select as mission field the same 110 m-long corridor (U-shape) used for collecting part (16%) of the Himax dataset images. The mission field

presents only static obstacles (e.g., walls, doors, and furniture), where we perform 25 tests, sweeping the v_{target} parameter. We employ a growing-step of 0.5 m/s, from v_{target} = 0.5 m/s to 2.5 m/s, testing each configuration 5 times. All these experiments are made by selecting the maximum performance SoC's configuration (FC@250 MHz, CL@175 MHz), able to deliver up to 12.8 frame/s inference.

Table 4.3. Evaluation of the flight time and average velocity when the drone flies through a 110 m long corridor.

v_{target} [m/s]	v_{avg} [m/s]	Average time [s]	Distance [m]	Success rate
0.5	0.51	216	110	**5/5**
1	0.98	112	110	**5/5**
1.5	1.72	64	110	**5/5**
2	1.96	56	110	4/5
2.5	**2.29**	**48**	110	1/5

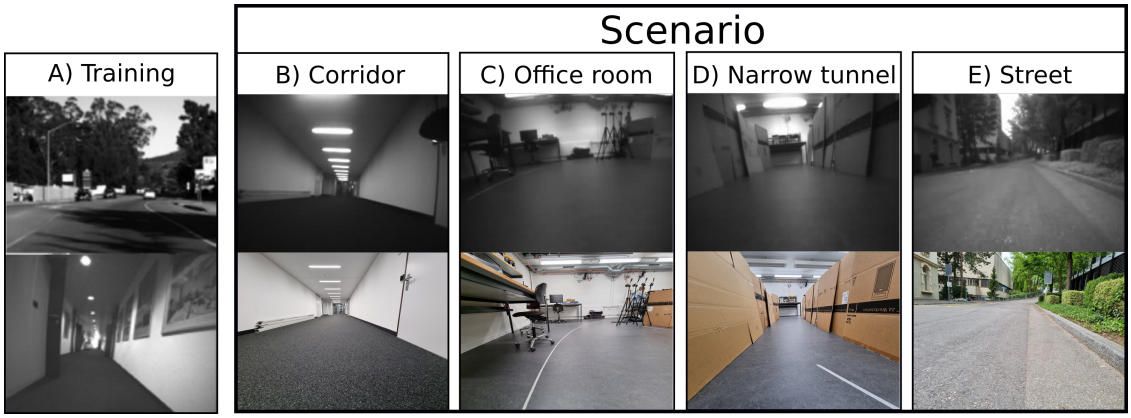


Figure 4.7. Samples of: A) the training images (both Udacity and Himax dataset); B) working-place corridor; C) office room with furniture; D) narrow pipeline-like tunnel; E) public street.

In Table 4.3, we summarize all the experiments for each v_{target} configuration, reporting the average flight time over the successful runs and highlighting in bold the peak performances. We achieve the maximum success-rate (5 success out of 5 tests, per configuration), with an actual mean flight velocity (v_{avg}) from 0.51 to 1.72 m/s. Increasing the v_{avg} to 1.96 m/s, lowers the the success-rate to 80%, defining the performance upper bound of our system, as increasing even further the v_{avg} to 2.29 m/s the success-rate quickly drops to 20%. With such high velocity also comes an increased acceleration the drone applies to reach the desired v_{target},

resulting in a high positive pitch. Therefore, the camera mostly captures the floor, which turns in a minimal time to react to the obstacles, flying at high speed. These results prove a superior performance compared to PULP-Dronet V1, which for the same flown distance reports an average velocity of 0.5 m/s. While PULP-Dronet V1 used a linear mapping between the probability of collision and the forward velocity, we extended this mechanism with a quadratic relation. This results in a more significant reduction in the forward velocity while approaching an obstacle, which gives the drone more time to steer when flying at high velocities (prior to steering) [4]. The main limitation of this experiment is that the corridor is "known" to PULP-Dronet V2 as the training set contains images of the corridor: in the next section, we explore the network's capability to generalize to never-seen-before testing environments.

Generalization capability

As the last part of our in-field evaluation, we present a set of functional experiments aiming at demonstrating the robustness of the PULP-Dronet V2 CNN, testing the closed-loop system in different deployment scenarios. As reported in Table 4.4, and showcased in Figure 4.7, we explore four different application scenarios, namely:

1. **corridor:** a working-place corridor that is not part of the Himax dataset;
2. **office room:** a room with tables and lab equipment;
3. **narrow tunnel:** a narrow pipeline-like tunnel ($\sim$5 m long and $\sim$1.2 m wide) made of cardboard;
4. **street:** a public street, with road signs and cars.

Orthogonal to the four scenarios, we also consider a second testing criterion: the presence/absence of obstacles in the path the nano-drone should follow. For the cases *corridor*, *office room*, and *street*, the obstacle is represented by a person that can be either standing still in the center of the path (i.e., static obstacle) or moving and crossing the trajectory of the drone or stopping in front of it (i.e., dynamic obstacle). Regardless of the type of obstacle, the goal is to adjust the moving direction, avoiding the obstacle. Instead, for the *narrow tunnel* case, we employ a small cardboard panel as an obstacle, still differentiating between a static and dynamic configuration. All these experiments refer to a setup with a drone's mean target velocity $v_{target} = 0.5$ m/s.

Considering the training datasets shown in Figure 4.7-A, the selected deployment

[4] As supplementary material, we make available video footage of one run, with $v_{target} = 2.0$ m/s, available at `https://youtu.be/41IwjAXmFQ0`.

fields introduce a significant difference between the visual cues present in the in-field images and those the CNN has been trained with. The main two sources of difference between the training and deployment can be ascribed to *i*) environmental conditions (e.g., absence of road lane signs in the street) *ii*) photometric/geometric differences between the cameras used to acquire the vast majority of the training dataset and the one available on the mission drone (e.g., field-of-view and resolution).

Table 4.4. In-field evaluation scenarios over multiple runs with and without obstacles (mean flight time : success-rate).

Scenario		**Obstacle**		
		None	Static	Dynamic
	1. Corridor	144 s : 6/6	137 s : 3/6	63 s : 4/6
	2. Office room	86 s : 5/6	78 s : 4/6	67 s : 5/6
	3. Narrow tunnel	12 s : 5/6	0 s : 0/6	19 s : 4/6
	4. Street	171 s : 6/6	148 s : 6/6	148 s : 6/6

In Table 4.4, we report the results in terms of mean flight time (across multiple runs) and success-rate, for each case. For scenarios 1, 2, and 4, we consider the test successful if the drone follows the path, with no crashes, for at least 60 s. Instead, for scenario 3, we define as success criteria the capability of the nano-drone to complete the exploration of the entire narrow tunnel. Among all successful cases, the mean flight time spans from 12 s to 171 s, for scenario 3 with no obstacles and scenario 4 with no obstacles, respectively.

The PULP-Dronet V2 sample application shows a high success rate for all configurations except for the *narrow tunnel* with static obstacles, in which case the nano-drone gets stuck in the obstacle proximity, slowly drifting towards it, until it suddenly crashes. As introduced in Section 4.1.1, the training datasets have disjoint labels, with the Udacity set providing only steering labels and both Bicycle and Himax sets coupled with only collision ones. Therefore, the actual tasks learned by the CNN can be defined as *"predict the presence of obstacles"* and *"predict the steering to follow the lane"*, but not explicitly *"predict the steering to prevent a collision"*. This limit of the Dronet CNN (all versions) could be mitigated in the future by either introducing a new training dataset providing both labels for each image sample or introducing an additional level of intelligence between the CNN and the low-level control.

However, this group of tests aims at assessing the generalization capability of the PULP-Dronet V2 baseline model without introducing any additional deep learning

technique, such as deep domain adaptation [93], dataset augmentation [19] or continual learning [94], as they would be out of the scope of this manuscript. All the nine configurations presented in this evaluation are provided with video footage of the experiments available at `https://youtu.be/Cd9GyTl6tHI`.

4.1.4 Conclusion

The recent progress in deep learning research opens new possibilities for using vision-based end-to-end neural networks to enable nano-drone autonomous navigation. The MCUs found in nano-drones have limited memory and computational resources, and therefore they are unable to run complex CNN models in their original form. However, the available solutions for complexity reduction of the CNNs used to facilitate navigation mainly involve hand-crafted modifications and typically require multiple iterations. This Section fills this gap by analyzing and integrating tools and methodologies that automate this optimize-and-deploy process, automating fine-grained hardware-aware tuning of the CNN. We perform an extensive experimental evaluation of the proposed flow, using a SoA CNN for autonomous nano-drone navigation [18], achieving $\sim 3{-}4\,\text{mJ/frame}$ inference and reducing the memory requirements by $2\times$ and improving the throughput by $1.6\times$ while preserving the same $\sim 90\%$ classification accuracy of the original implementation. Furthermore, we perform an in-field evaluation of the navigation capabilities of a nano-drone, considering the collision avoidance, steering capabilities, maximum flown distance, and generalization in never-seen-before environments. We record a maximum indoor flight distance of 110 m and an average velocity of 1.96 m/s, $4\times$ higher than our PULP-Dronet baseline. We foster the research community releasing as open-source our code and models: `https://github.com/pulp-platform/pulp-dronet`.

The contribution of the author in this Chapter have been the following:

- trainig of the CNN models;
- quantization of the CNN with the NEMO;
- evaluation of the CNNs on the datasets, before and after quatization;
- deployment of the CNN with the DORY;
- evaluation of the CNN's power consumption and the inference performance;
- state-of-the-art comparison with repsect to the CMSIS-NN library;
- power breakdown analysis of the nano-UAV.

4.2 A Sim-to-Real Deep Learning-based Framework for Autonomous Nano-drone Racing

Figure 4.8. Our nano-drone winning the IMAV'22 "Nanocopter AI Challenge."

Competitions have always been a catalyst for scientific and technological progress. As the *space race* was a driver to develop programmable computers and microchips, likewise, in recent years, autonomous drone racing pushed the development of cutting-edge navigation algorithms, including artificial intelligence (AI), running aboard unmanned aerial vehicles (UAVs) [26, 25, 16]. Competitions act as a proxy to improve UAVs' perception, planning, and control skills: from 2016 to date, the racetrack's average flight speed of autonomous drones increased from 0.6 to 22 m/s [35]. Eventually, these advancements positively impact a more comprehensive range of applications where robust, agile, and precise autonomous navigation is crucial, such as rescue missions [9] and human-robot interaction [19].

So far, these competitions have focused on micro-sized drones, i.e., ∼30 cm-wide robots capable of hosting powerful processors and rich sensors, *autonomous nano-sized drone racing* constitutes a newborn category employing palm-sized UAVs. The first competition of this kind was the "Nanocopter AI Challenge" hosted at the 13^{th} International Micro Air Vehicle conference (IMAV'22). This last-born class of robotic competitions poses a new challenge to roboticists due to the small size of nano-drones, i.e., 10 cm in diameter and a few tens of grams in weight. While nano-drone racing targets tasks similar to major competitions for bigger drones (e.g., AlphaPilot, IROS Drone Racing, etc.), their ultra-tiny form factor allows only minimal onboard resources, i.e., memory, computation, and sensors.

In the IMAV'22 competition, all challengers run their navigation algorithms on the same platform: a commercial off-the-shelf (COTS) Crazyflie 2.1 nano-drone (Figure 4.8) equipped with the AI-deck board featuring a GWT GAP8 System-on-Chip (SoC) [56] and a grayscale, low-resolution camera [24]. Compared to the typical processors found on micro-sized racing drones [25, 26], e.g., Nvidia Jetson Xavier, the GAP8 SoC has more than 1000× less compute power and memory. Despite this, the competition encouraged onboard computation by granting a 5× score multiplier while challenging the nano-drone with agile maneuvers to *i*) avoiding static and dynamic obstacles and *ii*) passing through a set of gates in a **never-seen-before** indoor arena. In fact, before the competition, no arena map or real-world dataset was released; participants could only access a photorealistic simulator.

This Section's main contribution is a thorough analysis and description of the strategy, methodology, and technical implementation we employed to win the IMAV'22 competition: a fully-onboard deep learning-based visual navigation framework trained only on simulation data. In detail, we present *i*) an exhaustive discussion of our strategy, which accounts for both the competition's guidelines and the nano-drone's limitations; *ii*) a convolutional neural network (CNN) for obstacle avoidance derived from the open-source *PULP-Dronet* [1] and trained only on simulation images; *iii*) mitigation of the *sim-to-real* gap [95] via aggressive photometric augmentation, label balancing, and comprehensive data generation; *iv*) three alternative *navigation policies*, which we characterize both in simulation and in the field.

Our final system, employing the best-performing navigation policy, ranked first among six contending teams at the IMAV'22 competition. In our best run, we scored 115 m of traveled distance in the allotted 5 min-flight, never crashing, dodging dynamic obstacles, and only using computational resources aboard our nano-drone. Our result demonstrates the effectiveness of the proposed sim-to-real mitigation strategy, our implementation's robustness, and our rationale's soundness, and by sharing our insights, we aim to provide the research community with a solid groundwork for the evolution of this field.

4.2.1 The IMAV'22 Nanocopter AI Challenge

The competition: the metric to assess each team's score is the distance traveled within the mission area (the 8×8 m green square shown in Fig. 4.9-A) within the allotted time (5 min), employing a Crazyflie 2.1 nano-drone. This distance is measured with a motion capture system that ignores the part of the trajectory lying outside the mission area. Participants can choose between two coefficients of difficulty, one accounting for *environmental complexity* (α_{env}) and the other for *computational resources* (α_{comp}). Additionally, in the arena, there are two rectangular gates through which the nano-drone can fly; every time the drone passes

through a gate, the distance is increased by an additional 10 m.

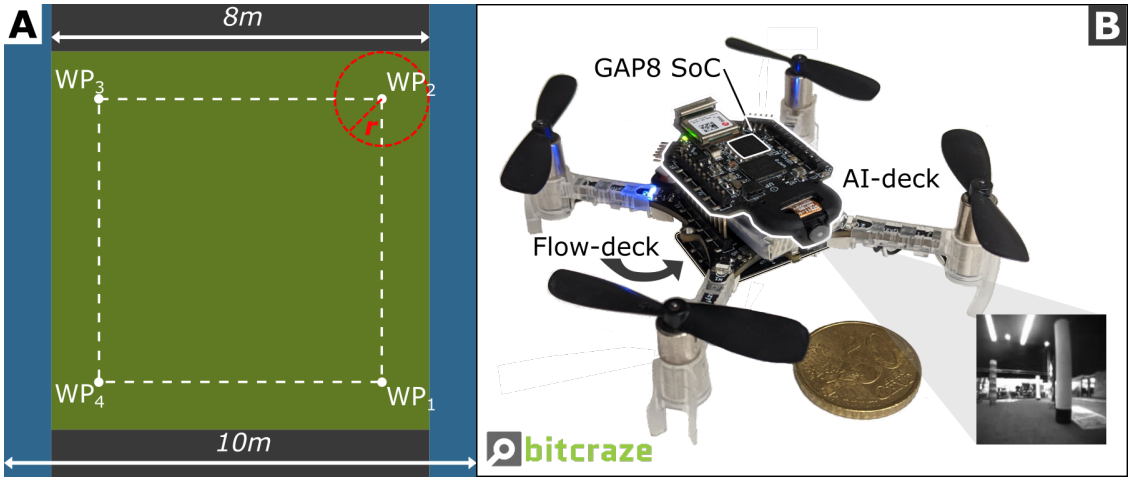


Figure 4.9. A) the 10×10 m competition arena. B) the robotic platform.

Participants can choose among three levels of environmental complexity: only static gates ($\alpha_{\text{env}} = 1\times$), static gates and obstacles ($\alpha_{\text{env}} = 5\times$), and static gates and dynamic obstacles ($\alpha_{\text{env}} = 10\times$). If the nano-drone uses remote computation (i.e., Wi-Fi-connected commodity laptop), the α_{comp} is 1×; for onboard computation, instead, it is $\alpha_{\text{comp}} = 5\times$. The final score is calculated as follows:

$$Score = (S_{\text{dist}} + 10 \cdot S_{\text{gates}}) \cdot \alpha_{\text{env}} \cdot \alpha_{\text{comp}} \tag{4.2}$$

where S_{dist} is the total traveled distance and S_{gates} is the number of times the drone passes through the gates. Time counting is never stopped (except for battery swaps): crashing, re-flashing, rebooting, components replacements, and other flight interruptions are allowed, but they will ultimately penalize the score. The teams can choose the starting position of the drone, but after a flight interruption, subsequent take-offs must occur near the location where the flight was interrupted.

Robotic platform: all teams competed with the same robotic platform comprising a COTS Bitcraze Crazyflie v2.1 drone and two additional onboard modules, as shown in Fig. 4.9-B. The drone is a modular 27 g 10 cm quadrotor integrating sensing, communication, and actuation subsystems with an STM32F405 MCU for sensing, state estimation (through an extended Kalman Filter) and control. The UAV can fly up to ∼7 min employing a 250 mA h battery. The first additional module is the Flow-deck, a small (21×28 mm) printed circuit board (PCB) that provides additional sensors to improve the state estimation: a VL53L1x Time-of-Flight distance sensor for measuring the height from the floor and a PMW3901 low-resolution (35×35 px) optical-flow camera.

The second expansion board employed is the AI-deck, a PCB for vision-based onboard processing. The module embeds *i*) an Himax HM01B0 low-power (∼4 mW) QVGA monochrome image sensor camera, *ii*) a NINA-W102 MCU supporting Wi-Fi and Bluetooth communications, and *iii*) GAP8, an ultra-low-power (ULP) SoC capable of efficiently managing computation-intensive workloads by employing hardware-enabled (8-core cluster of processing cores) parallel programming paradigms. The SoC also features standard interfacing peripherals (SPI, I2C, UART) as well as different hierarchically organized SRAM memories (L1:64 kB L2:512 kB).

Simulator: the IMAV challenge organizers provided all teams with a simulator developed and released by Bitcraze, based on the open-source *Webots* robot simulator. The simulator includes a simple PID controller for the attitude control of a Crazyflie quadcopter. It emulates the 87° field of view of the drone's Himax camera, and it provides a photorealistic world model of the competition arena. Several obstacles in the simulator resemble the ones of the IMAV challenge, including an orange pole; an orange gate; three panels with a width of 1 m, 1.5 m, and 3 m, respectively; curtains; carpets; nets.

Our strategy: given IMAV's rules, to maximize the overall score, the nano-drone should: *i*) address two concurrent complex vision-based tasks, i.e., *obstacle avoidance* and *gate-based navigation*; *ii*) rely only on onboard computation; *iii*) operate in the most challenging environment. Relying on off-board computation can ease the computational burden. However, this approach reduces the overall score due to the lower coefficient of difficulty and decreases the drone's reactiveness due to the additional Wi-Fi communication latency. Instead, individually addressing even one of the two tasks would represent a significant step forward in the SoA. Neither simulation-based obstacle avoidance nor gate-based navigation is still demonstrated in a challenging autonomous nano-drones race. At the same time, as previously discussed, the computational and memory limitations of embedded platforms such as GAP8 still constitute a strong holdback against deploying multiple tasks simultaneously. Therefore, for the IMAV'22 competition, we prefer to address only one of these tasks while challenging our system with fully onboard computation ($\alpha_{\text{comp}} = 5\times$) and maximum environmental difficulty ($\alpha_{\text{env}} = 10\times$).

To choose which task to perform, we consider two ideal nano-drones A and B flying at a mean speed of 1.5 m/s in a thought experiment. A performs obstacle avoidance in the most complex environment ($\alpha_{\text{env}} = 10\times$), while B addresses gate-based navigation in an obstacle-free scenario ($\alpha_{\text{env}} = 1\times$). In a 5 min flight, the A system would score 7500 points. Assuming the two gates are 3 m apart, the B system would travel 750 m, passing through the gates 250 times, leading to a final score of 3250 points. Therefore, we believe that with the rule set of this competition, an autonomous nano-drone that quickly explores its surroundings and reliably avoids

collision with dynamic obstacles (scenario A) has the potential to mark a higher score than a system designed for gate-based navigation in the simpler scenario B.

4.2.2 Navigation policies

Deep Neural Network: the neural network used in this Section is derived from the open-source PULP-Dronet CNN [1]. We keep the same network topology consisting of three consecutive residual blocks (ResBlocks). Each ResBlock comprises a primary branch that executes two 3×3 convolutional layers and a parallel by-pass employing a 1×1 convolutional layer. Unlike the previous work, whose outputs consisted of a single collision probability and a steering angle, our CNN features three collision probabilities by splitting the input image horizontally in three 54×162 px left, center, and right portions of the field-of-view (FoV). We exploit only simulated data to train the network, and the data collection procedure is detailed in Sec. 4.2.3. After training the network, we apply fixed-point 8-bit quantization to its weights to: *i*) reduce the CNN's memory footprint by $4\times$, resulting in a size of 317 kB, and *ii*) to enable optimized 8-bit fixed-point arithmetic, resulting in a throughput of 30 frame/s when deployed on GAP8. We define three navigation policies that rely on the CNN outputs: Baseline, Policy 1, and Policy 2.

Baseline: for each input image, the drone flies towards the direction with the lowest collision probability among the three CNN's outputs. In case all three probabilities are higher than a given threshold $\mathcal{T}_h$, the drone spins in place by 180° ± a random angle in the [0°, 30°] range. To generate the training labels for this model, only actual objects in the environment (i.e., panels, cylinders, walls, gates, etc.) are considered obstacles.

Policy 1: same as the baseline policy, with the difference that the training labels are generated considering as an obstacle also the ground outside the mission area, i.e., black and blue stripes in Fig. 4.9-A, as well as surrounding walls.

Policy 2: the three probabilities of collision are trained as in Policy 1, and if at least one of them is higher than a given threshold $\mathcal{T}$, the drone will behave as in the previous two policies. Otherwise, if all probabilities of collision are lower than $\mathcal{T}_h$, the drone will try to reach one of the four waypoints (WPs) defined as the corners of a square inscribed in the mission area, $\text{WP}_{1,2,3,4}$ in Fig. 4.9-A. This *WP-based navigation* mechanism leverages *i*) the *a priori* knowledge of the take-off point, i.e., we can choose the take-off point/orientation of the drone, for example, in WP_1; and *ii*) the onboard visual-inertial state estimation which provides the relative position of the drone w.r.t. the take-off point. Then, when the drone is in the WP-based navigation mode, it will try to visit the WPs in a predefined cyclical order; for example, $\text{WP}_{1,2,3,4}$ produces a counterclockwise motion, and $\text{WP}_{4,3,2,1}$ a clockwise one. Lastly, to mark a WP as "visited," we define a circular area of radius r centered in each WP (in red in Fig. 4.9-A): when the drone enters such an area, we mark the

corresponding WP as visited and continue with the next one. Whenever all three collision probabilities are higher than the threshold $\mathcal{T}_h$, we invert the sequence of the targeted waypoints, alternating a clockwise and counterclockwise direction. This mechanism also serves as an escape strategy when a WP is unreachable.

Rationale discussion: autonomous drones flying with a priori knowledge of the environment, i.e., a map, are highly effective as we can plan for the best trajectory. Unfortunately, the ultra-constrained resources (memory, computation, and sensors) aboard a nano-drone make this map-based navigation unfeasible [82] or extremely limited [32]. However, in the IMAV'22 competition, we can exploit coarse-grained information, as we know the size of the mission area, the colors of the ground, and – up to some degree – the shape/texture/colors of the objects thanks to the simulator provided by the organizers. Since obstacles and gates are more likely to populate the inner part of the flying area rather than its borders, Policy 2 tries to maximize the likelihood of a straight obstacle-free path by positioning the WPs at the edges of a square.

Therefore, ideally, we should place our four WPs precisely on the edges of the green square in Fig. 4.9-A. However, the unavoidable noise/drift of the simple state estimation aboard our nano-drone would often drive the vehicle outside the mission field, negatively impacting the final score. For this reason, we introduce the WPs as the edges of an inner square within the available 8×8 m mission area.

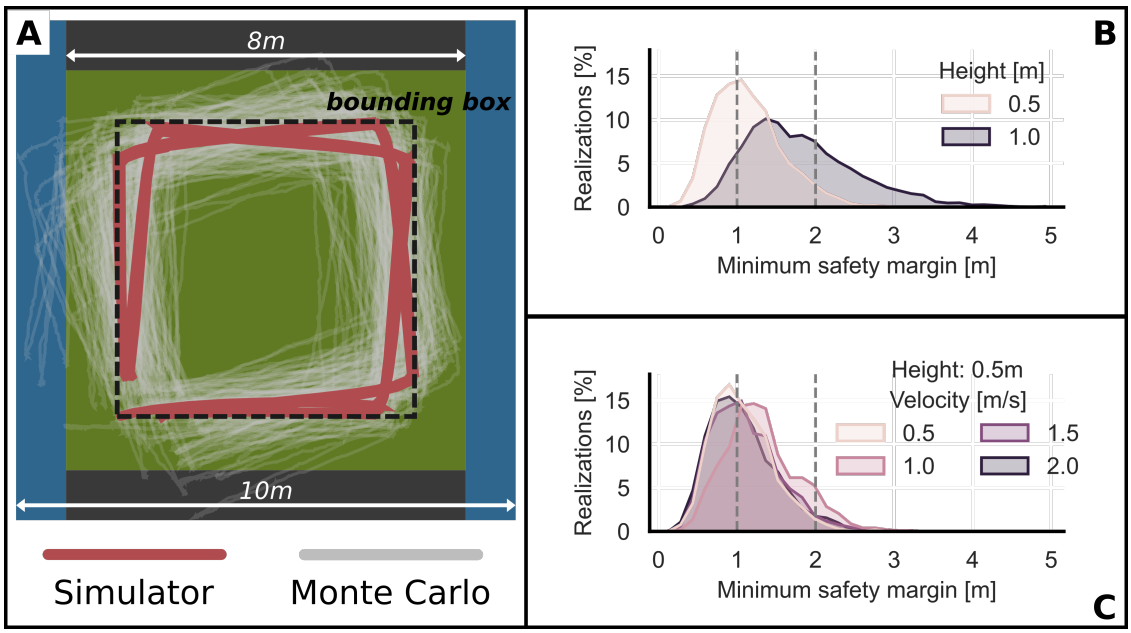


Figure 4.10. A) 1024 simulated Monte Carlo trajectory realizations B) minimum safety margin w.r.t. height, C) minimum safety margin w.r.t speed, having fixed the height to 0.5 m.

State estimation analysis: we characterize the position and orientation error of the Crazyflie's state estimation subsystem. This allows us to determine a margin between the WPs and the corners of the mission area that *i*) keeps the drone flying close to the edges of the area and *ii*) minimizes the risk that state estimation drift drives the drone out of the area. We start by collecting real-world flight data, which we use to build a statistical model. We rely on a setup similar to IMAV competition with artificial turf on the floor, albeit over a smaller 5 ×6 m flying area. We record the drone's movements with a motion-capture system and the drone's onboard state estimation, both at 100 Hz. We perform runs of 5 min each, in which the drone flies using a random walk policy while constrained through the mocap into an empty squared area – smaller than our 5×6 m arena. We perform 16 flights: two runs for each configuration combining two flight heights, i.e., 0.5 and 1.0 m, and four mean speeds, i.e., 0.5, 1.0, 1.5, 2.0 m/s.

Similarly to [96, Sec. 5.2.4], we quantify the state estimation error by modeling the uncertainty on incremental state estimation updates. We sample many ten-second time windows from the collected data, considering the relative pose of the drone at the beginning and the end of each window. We measure the estimation error separately for each component of the relative pose: x, y, and yaw. For each component, we fit a Gaussian distribution to the errors measured over many windows, then we rescale the distribution to represent the error accumulating in 1 second.

Then, in the Webots simulator, we collect eight runs, i.e., all combinations of heights and speeds, of 5 min each (red lines in Fig. 4.10-A). The drone follows a squared trajectory on a 6 ×6 m square, which is intuitively a safe configuration. We invert the flown path for each lap by alternating clockwise/counterclockwise directions to ensure that any systematic errors in yaw estimation (e.g., a consistent under/over-estimation of the amount of rotation) cancel out rather than accumulate. We use these eight flown trajectories to build a *bounding box* enclosing them (dashed black line in Fig. 4.10-A). We apply the statistical error model to corrupt the ideal squared trajectory, computing 1024 Monte Carlo (M. C.) realizations of the drone's state estimation (pale gray lines in Fig. 4.10-A). Then, only considering the portion of the M. C. trajectories outside the bounding box, we measure the maximum distance between them and the box itself. This measure gives us a statistical *minimum safety margin* distribution that will likely constrain the flight within the desired flight area.

Fig. 4.10-B shows (average on all speeds) that flying at 0.5 m height requires a lower safety margin than at 1.0 m, centering the distribution at 1.08 m and 1.56 m, respectively. Therefore, we select 0.5 m as our target height, and we focus only on this configuration in Fig. 4.10-C, where we explore the four velocities. Among all velocities, 42% of the 1024 realizations never exceed a safety margin of 1 m,

while 95.3% satisfy a margin of 2 m. Furthermore, the median time spent outside the 1 m margin is 2 s (95^{th} percentile: 31 s) per realization. Finally, of the 4.7% of realizations that cross the margin of 2 m, the median number of crossings is only one per realization (95^{th} percentile: 5). These findings confirm that positioning our WPs at the edges of a 6×6 m square will *i*) keep the drone in the desired 8×8 m mission area on average for 97.6% of the 5 min run's time; and *ii*) cause the drone to exceed the maximum 10×10 m space, *i.e.,* crash, only with 4.7% probability and only once per run. Finally, the four flight speeds analyzed do not significantly impact the safety margin requirements, which leads us to use the highest speed: 2.0 m/s.

4.2.3 CNN training and deployment

Dataset collection: we use the Webots simulator, shown in Fig. 4.13-A, to collect a dataset of images for our CNN. To automatically generate the labels for each image, we extended the simulator's capabilities by introducing the generation of depth and segmentation frames from the camera. During the image collection, we use a flight height of 0.5 m, see Sec. 4.2.2. Furthermore, to mimic the dynamic effects of an actual UAV flight, we add random variations for pitch, roll, and yaw in a $[-5°; +5°]$ range and for the height in a $[0.45; 0.55]$ m range.

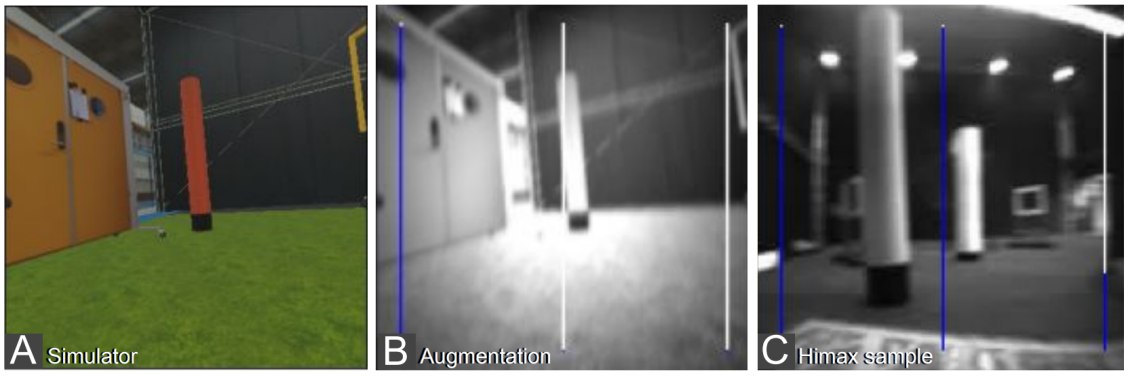


Figure 4.11. Images: A) from the simulator, B) after augmentation, C) Himax camera sample collected in the IMAV arena. The three blue bars in images B-C) represent the three collision probabilities predicted by our network.

We collect images with a 324×324 px resolution and 87° FoV, the same as our Himax camera. For each image, we save per-pixel depth and segmentation masks, which we exploit for labeling: we divide the camera FoV into three vertical portions of 108×324 px, and for each portion, we set a collision label = 1 if the 10% of the pixels belonging to obstacles has distance $\leq$2 m, 0 otherwise. We consider all kinds of objects as obstacles, *including gates* and excluding carpets. We collected 41 k images in 3 simulated scenarios to ensure labels balancing and uniform sample

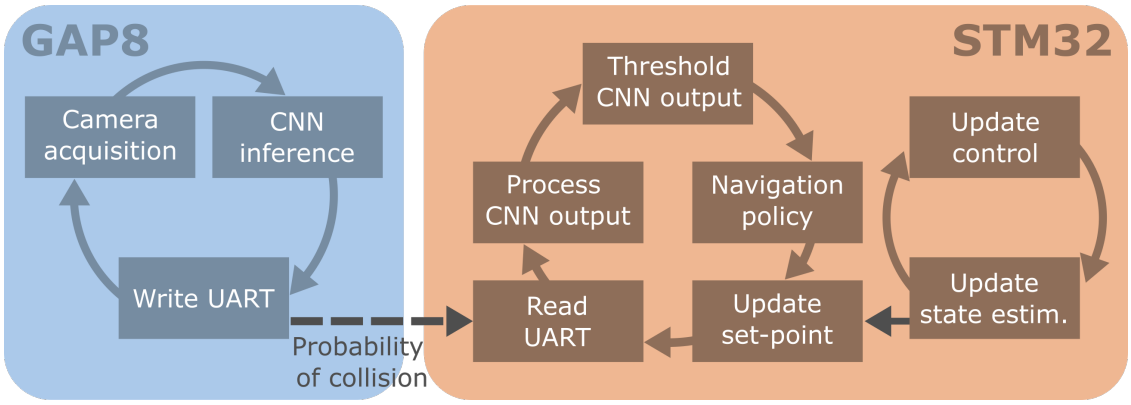


Figure 4.12. The drone's state machines mapped on the MCUs available aboard.

distribution. 10 k images are collected by spawning the drone randomly across the arena populated by obstacles, and 21 k images are acquired by flying around each obstacle in the simulator. This last group comes from a "360 scan" of each obstacle, which we equally split between images with an empty background and a populated one. Finally, 10 k images are taken from the drone flying along a square trajectory, 50 cm within the green flying area edges, equally split between clockwise and counterclockwise flight directions. Eventually, we split each portion of the dataset using 70% of the images for training, 10% for validation, and 20% for testing.

Photometric augmentations: as the simulated images are noise-free 3D renderings of the scene, we introduce an aggressive data augmentation pipeline to bridge the appearance gap with real-world Himax images. We randomly perturb the simulated images to reproduce several real-world image artifacts: motion blur, Gaussian blur to simulate lens defocusing, Gaussian noise, and exposure changes (gain, gamma, dynamic range, and vignetting). Finally, we convert images to grayscale and resize them to 162×162 px, as our Himax camera.

Fig. 4.11-A-B and the supplementary video show the result of this photometric pipeline, while Fig. 4.11-C displays a real drone frame. We assess the impact of the photometric augmentations on CNN accuracy by exploring all eight combinations of exposure, blur, and noise augmentations. To evaluate the accuracy, we collected 2200 real-world images from our 5×6 m indoor flying arena, shown in Fig. 4.13-B. The results in Tab. 4.5 shows that enabling all augmentations scores the highest accuracy, surpassing the non-augmented model by 17%.

Navigation policies implementation: Fig. 4.12 shows the state machines mapped on the two MCUs available aboard our UAV. GAP8 implements a loop for *i*) image acquisition, *ii*) CNN inference, and *iii*) UART transmission of the 3 CNN collision

Table 4.5. Neural network accuracy by photometric augmentation method: Noise (N), Blur (B), and Exposure (E).

Aug.	**N**	***None***	**B**	**B+E**	**N+B**	**E**	**N+E**	**N+B+E**
Acc.	55%	56%	58%	62%	64%	68%	69%	72%

probability outputs, i.e., {left, center, right}, to the STM32. The STM32 instead implements two loops for high and low-level control of the drone, respectively. The high-level control loop *i*) reads the CNN's output from UART, *ii*) converts the 8-bit fixed-point CNN's outputs ([0,255] range) to `float32` numbers ([0,1] range) and applies a low pass filter, *iii*) thresholds the three probabilities of collision to$\mathcal{T}_h = 0.7$, getting values $\in$ {0,1}, *iv)* applies the navigation policy to calculate the next set point, and *v)* updates the low-level controller. Instead, the low-level control loop, running at 100 Hz, updates the state estimation through the eKF and applies a cascade of PID controllers to reach the set point pushed by the high-level control loop.

All three navigation policies described in Sec. 4.2.2 output a {speed, yaw rate} tuple as a set point. For all policies, the forward speed is inversely proportional to the *center* collision probability. The yaw rate is set to -90 or +90 deg/s when the right or the left collision probabilities get over the threshold th, respectively. If all three collision probabilities exceed the threshold $\mathcal{T}_h$, the drone spins in place as described in Sec. 4.2.2. Conversely, the three policies act differently when all collision probabilities are zero: Baseline and Policy 1 command the drone to fly in a straight line, while Policy 2 activates the WP-based navigation mode, heading the drone to the next WP.

4.2.4 Experimental results

In-simulator evaluation

We evaluate our three navigation policies with the Webots-based simulator, sample picture shown in Fig. 4.13-A. We rely on an ideal flight controller inside the simulator, meaning *i*) no state-estimation drift and *ii*) exploiting depth maps for ideal obstacle avoidance. We tested nine configurations combining the three control policies introduced in Sec. 4.2.2 and at three speeds (1, 1.5, and 2 m/s). For all the tests, we used seven obstacles: two orange poles, two gates, and three panels with a width of 1, 1.5, and 3 m, respectively. We run ten experiments of 5 min each for each configuration, varying the obstacles' initial position but keeping it consistent among the nine configurations. To emulate the dynamic obstacles of the challenge, we move one obstacle every 30 s. We compare the policies based on the distance traveled on the green turf.

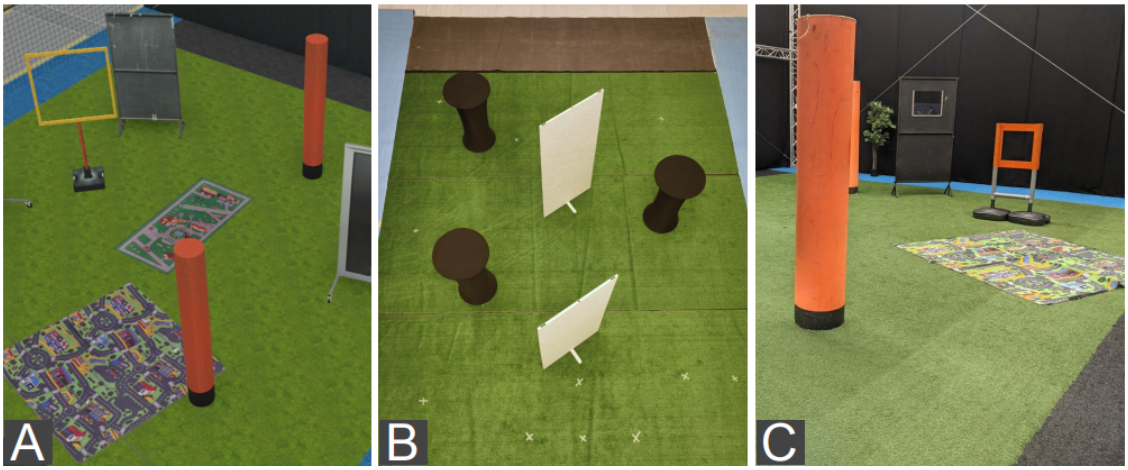


Figure 4.13. Three environments: A) the *Webots* simulator, B) our indoor 5 ×6 m arena, and C) the 8×8 m competition arena.

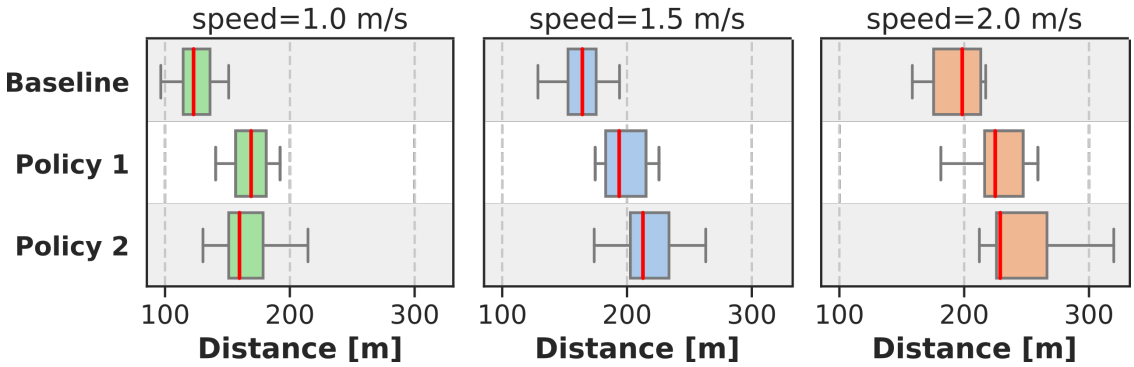


Figure 4.14. Median traveled distance (10 runs) in 5 min flights on the simulator of our three navigation policies at 1.0, 1.5, and 2.0 m/s.

Fig. 4.14 shows that the baseline policy (which does not consider the floor outside the green turf as an obstacle) is consistently worse than the others, spending 16% of the time outside the mission area. Conversely, Policy 1 and 2 detect the external area as an obstacle, spending only 1% and 0.8% of the time on the external area, respectively. Both Policy 1 and 2 benefit from higher flight speeds due to the simulator's perfect sensing preventing collisions. At the lowest speed configuration of 1 m/s, Policy 2 achieves a slightly lower median distance than Policy 1, 165 m vs. 169 m. Instead, for higher speed configurations, the median distance of Policy 1 is slightly lower than Policy 2: 217 m and 246 m at 1.5 m/s, and 193 m and 225 m at 2 m/s, respectively. Since both Policy 1 and Policy 2 score a similar traveled distance, we push forward our analysis by deploying and testing both of them in the field.

State-of-the-Art comparison

Since our CNNs are inspired by the PULP-Dronet [1], as well as many other teams at the competition, we present a thorough comparison of our system against a *vanilla* PULP-Dronet and a fine-tuned version of it. Given the different outputs between our CNN and the original one, we adapt our navigation Policy 1 (no WPs) to the PULP-Dronet baselines. We use the PULP-Dronet steering output, as in its original implementation, by converting it in a yaw-rate for the controller in case of a probability of collision lower than 0.7, while the drone rotates 180°otherwise. For the fine-tuned version, we use the same dataset collected in simulation to train our models. The single collision label of PULP-Dronet is matched to the central probability of our models, while the steering angle reflects the direction with the lower probability of collision among the three probabilities: left, center, and right.

The results in Fig. 4.15 are collected in simulation, where we run ten 5-min tests for each model at three target flight speeds (i.e., 1, 1.5, and 2 m/s). The vanilla PULP-Dronet marks the lowest score in all configurations, 84–114% less than our system, while the fine-tuned PULP-Dronet reduces this gap with a performance reduction of 28%–43%, depending on the flight speed. Our network scores a median distance traveled of 171, 199, and 228 m at 1, 1.5, and 2 m/s, respectively, showing the benefit of our architectural changes and training methodology.

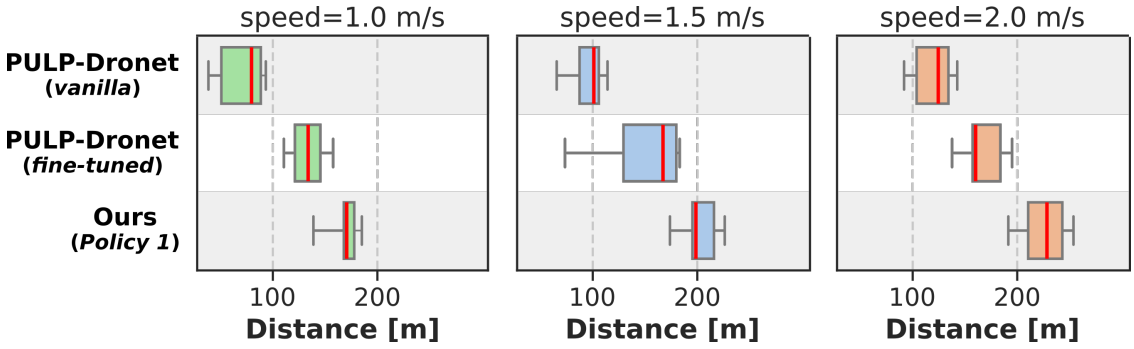


Figure 4.15. Median traveled distance (10 runs) in 5 min flights on the simulator comparing our work to the SoA PULP-Dronet [1] at 1.0, 1.5, and 2.0 m/s.

In-field evaluation

We evaluate Policy 1 and Policy 2 in our 5×6 m indoor flying arena, testing them with two target speeds, 1.5 and 2 m/s, resulting in four configurations. For each configuration, we perform 5 runs of 5 min each. We use three black cylinders and two white panels as obstacles, as shown in Fig. 4.13-B. We replicate the competition's conditions, e.g., periodically moving the dynamic obstacles (every ∼30 s), and without stopping the time count if the drone crashes. As shown in Fig. 4.16-A,

Policy 1 scores a median 45 and 65 m, while Policy 2 marks a median 91 and 92 m, while flying at 1.5 and 2 m/s, respectively. The improved performance of Policy 2 derives from the *WP-based navigation*, which provides the drone with boundaries, limiting the time spent in potentially dangerous areas outside the mission field. A sample run of Policy 2 is shown in Fig. 4.16-B. For the IMAV competition, we choose to use the Policy 2 at 1.5 m/s for the first run (conservative) and push our system to the 2 m/s limit in our second attempt.

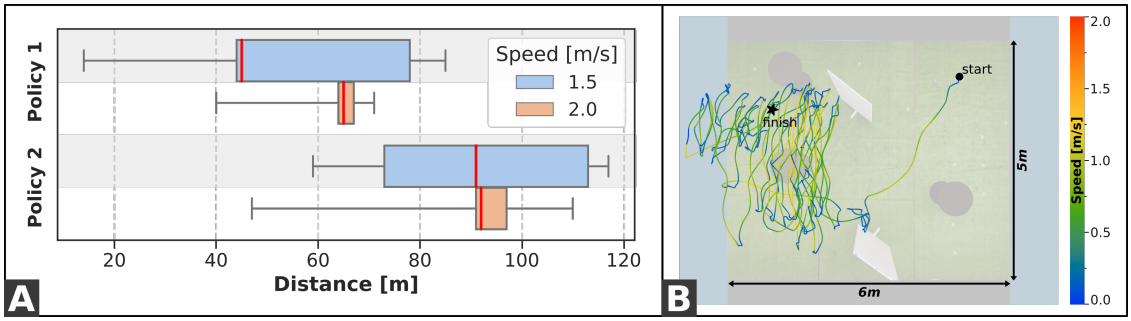


Figure 4.16. In-field testing, A) median traveled distance (5 runs) in 5 min flights of our two navigation policies at 1.5 and 2.0 m/s. B) Sample run of Policy 2 with a $v_{target} = 1.5$ m/s, scoring 117 m of traveled distance.

The nanocopter AI challenge

The arena for the competition was a 10×10 m area, represented in Fig. 4.9-A, where only the 8×8 m green surface is considered for the final score. The arena had ten objects (Fig. 4.13): four orange poles, two black panels, two flags, and two gates. The competition allows each team to perform two runs of 5 min each, having only the best one considered in the final leaderboard. Six teams participated, and the video recording is available online[5]. For each team, Tab. 4.6 summarizes the distance traveled, the number of gates passed, and the final score computed according to Eq. 4.2.

The last team achieved a traveled distance of 9 m in an obstacle-free environment while employing an off-board (i.e., WiFi-connected laptop) color segmentation algorithm. The 5th ranked team tackled an obstacle avoidance task using an off-board CNN for monocular depth estimation. They exploited the depth to control the drone's forward speed and steering angle commands, ultimately reaching a traveled distance of 67 m. The 4th-classified team proposed a system that passed through four gates, i.e., the highest number of traversed gates among all teams. They tackled gate-based navigation with an off-board color segmentation visual pipeline.

[5] `http://youtu.be/WaDU4I2TImA`

Table 4.6. The final leaderboard of the "Nanocopter AI challenge."

Team	Processing	Obstacles	[m] / Gates	Score
PULP (ours)	Onboard	Dynamic	115 / 0	5750
Black Bee Drones	Onboard	Static	81 / 0	2015
SkyRats	Onboard	Dynamic	29 / 1	1945
CVAR-UPM	Off-board	Dynamic	31 / 4	702
CrazyFlieFolder	Off-board	Dynamic	67 / 0	666
RSA	Off-board	No	9 / 0	9

However, their final traveled distance of 31 m was penalized by numerous crashes, wasting precious time for resuming the flight.

The team ranked 3$^{\text{rd}}$ used an onboard vanilla PULP-Dronet for obstacle avoidance, which led to a conservative ∼0.1 m/s average speed on their best 5-min run (*run 1*). Nevertheless, they managed to pass through one gate while crashing multiple times (up to five in *run 2*), resulting in a traveled distance of 29 m. The second classified team exploited only onboard computation in a static obstacles environment. Their navigation strategy limited the exploration to a small obstacle-free part of the environment, leading to a total distance of 81 m.

We tackled the most challenging scenario, i.e., dynamic obstacles, with only onboard computation (i.e., no WiFi-connected laptop), while leveraging our Policy 2, described in Sec. 4.2.2. In our two runs, we set a maximum target speed of 1.5 2.0 m/s, which resulted in a traveled distance of 115 and 97 m, respectively. During the first and best run, we never crashed for the entire 5-minute flight, resulting in the winning score of 5750 points, i.e., almost 3× more than the second-classified team. Fig. 4.17 summarizes our two runs, reporting a top-view of the arena where flight trajectories are shown with their mean average speed.

4.2.5 Conclusion and future work

We present the deep learning framework for visual-based autonomous navigation aboard nano-drones, winning the "IMAV'22 Nanocopter AI Challenge" drone race. Our system combines a CNN for obstacle avoidance, trained only in simulation, a sim-to-real mitigation strategy, and a navigation policy, defining the drone's control state machine. Our system scored 115 m of traveled distance at the competition while coping with static and dynamic obstacles. As we focused on the most challenging obstacle avoidance task while leaving out the gate-based navigation task, future work will address developing lightweight perception modules for both tasks. Nevertheless, our system marks the SoA being the first example of an autonomous

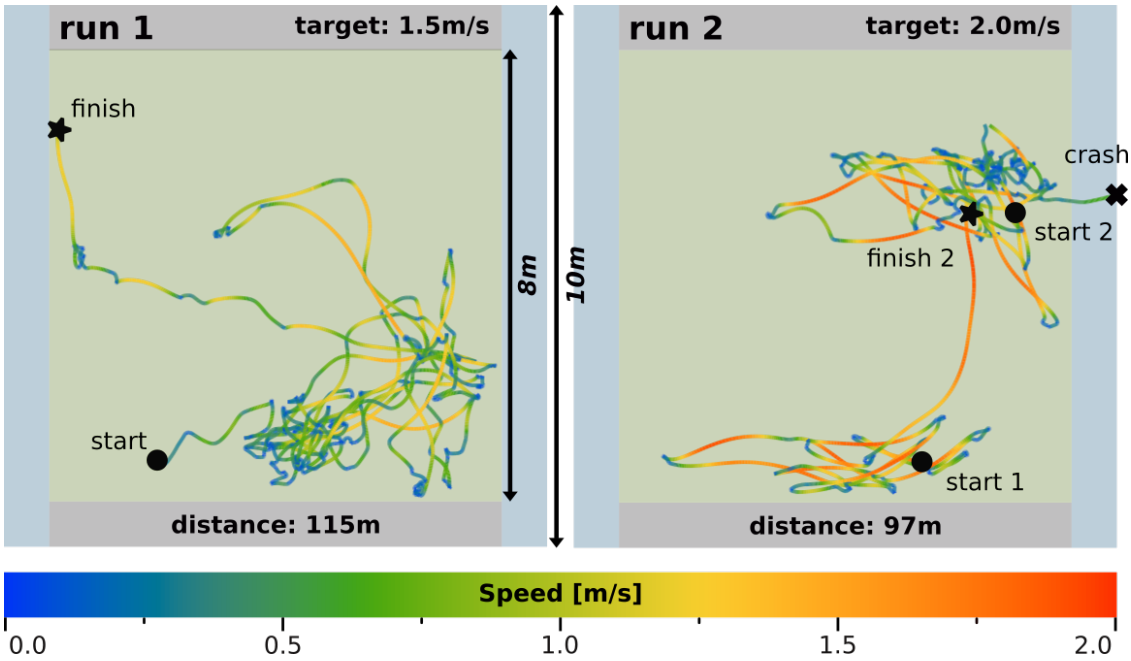


Figure 4.17. Trajectories of our two runs during the IMAV'22 race: run 1 ranked our team 1^{st} at the "Nanocopter AI challenge."

nano-drone completing its mission (5 min flight, at 1.5 m/s with no crashes) in a challenging never-seen-before race environment. Additionally, any bigger robot can exploit our lightweight yet accurate and reactive perception, freeing a vast amount of computational resources and memory that can be allocated to tackle additional tasks. By sharing our knowledge, we foster future research by providing a solid foundation in the newborn field of nano-drone racing.

Chapter 5

Shrinking NNs to enable multi-tasking AI on nano-UAVs

In Chapter 4, we demonstrated that executing individual AI tasks, such as visual-based autonomous navigation, is feasible on autonomous nano-UAVs. However, the computational and memory burden for enabling AI multi-tasking perception is still too high to be pursued on constrained MCUs. Therefore, this chapter aims to minimize the onboard intelligence workload needed for UAVs' autonomous navigation, freeing up enough resources to execute multi-perception intelligence tasks aboard a nano-drone. To do so, we first present a general methodology for evaluating deep learning models over parametrization (in terms of size and complexity). This methodology involves analyzing overfitting behaviors and examining the sparsity of Convolutional Neural Networks (CNNs) neurons. As an example, we apply this methodology to the SoA PULP-Dronet CNN introduced in Chapter 3, studying the various trade-offs between the number of channels, pruning of inactive neurons, architecture modifications, and accuracy. We ultimately demonstrate the effectiveness of our method by introducing novel squeezed CNN models called Tiny-PULP-Dronets, which are up to 50× smaller and 8.5× faster than the baseline CNN when running on the same processor. Nevertheless, we show that these models maintain the regression and classification performance when validated on the testing set. These networks leave sufficient memory and compute headroom for onboard multi-tasking intelligence, even at the nano-sized scale.

Before testing such networks in the field, we want to further improve the visual-based CNN for visual-based navigation by addressing the shortcomings of PULP-Dronet, introduced in Chapter 3. This CNN shows limited performance when the nano-UAV needs to navigate around static obstacles, therefore restricting its applicability in real-world scenarios. This limitation depends on the original CNN's

dataset construction, which combines data from different robotic domains with disjoint training labels to train the two CNN outputs, i.e., the collision avoidance and steering outputs. Therefore, we introduce a dataset-collecting methodology for collecting unified collision avoidance and steering information directly on the nano-UAV, and we use this methodology to collect a new dataset of 66 k images. Second, using this dataset, we introduce end-to-end training and deployment of a new family of CNNs, called PULP-Dronet-V3, that *i*) enable visual-based static obstacle avoidance and autonomous navigation suited to nano-drones, therefore improving the main navigation limitation of the SoA CNN (Chapter 3), and *ii*) we expand the investigation of the CNNs' memory footprint and workload minimization for autonomous nano-UAV navigation by studying multiple CNNs' architectures. Finally, we present an extensive in-field validation of the resulting tiny CNN for autonomous navigation on nano-UAVs, studying how the reduced workload affects navigation accuracy. We set up a challenging scenario consisting of a narrow corridor with four static obstacles and a 180° turn, and the new tiny CNN we designed shows the ability to successfully navigate with a 100% success rate through this scenario at a maximum target speed of 0.5 m/s. At the same time, the SoA un-pruned CNN consistently fails.

As a result, the CNN that resulted from our shrinking methodology only uses a fraction of the computational resources available on nano-UAVs: we obtained a 2.9 kB model achieving 139 frame/s throughput. This CNN still provides reliable navigation skills, which is the pivotal requirement for any IoT node not to crash when moving within an environment. In this way, we pave the way towards embedding multiple intelligence tasks on this nano-sized class of vehicles, as we can build up more intelligence on top of this optimized CNN for autonomous navigation.

5.1 Squeezing Neural Networks for Faster and Lighter Inference

With their sub-10 cm diameter and tens of grams in weight, nano-UAVs are agile and highly versatile robotic platforms, employed in many use cases where small size is crucial. From aerial inspection in narrow and dangerous places [4] to close-proximity human-robot interaction tasks [19], nano-UAVs are the ideal candidate platforms. Similar to other autonomous robots, also nano-UAVs navigate thanks to the interaction of three key sub-systems [97]. The *state estimator* determines the current state of the system. Then, what we call the onboard *intelligence* is responsible for solving the decision-making problem of choosing the next target state. Finally, the *control* part brings the system from the current state to the target one. Our work focuses on the intelligence, where the limited payload and power density of nano-drones clash with the onboard execution of complex real-time algorithms and even more with multi-tasking vision-based solutions.

Therefore, our goal is to minimize the onboard intelligence workload. This scenario allows to optimally balance between *i)* enhancing the UAV's reactivity with an increased throughput in the decision-making process [98], and *ii)* freeing up resources for the execution of multi-perception intelligence tasks aboard our nano-drone. These features enable nano-UAVs to tackle more complex flight missions. For example, autonomous aerial cinematography and high-speed drone racing require a combination of multiple tasks running concurrently aboard [12, 25], including visual-inertial odometry, object detection, trajectory planning, environment mapping, and collision avoidance. Ultimately, we could significantly push small-size robotic platforms towards the biological levels of intelligence [27].

To date, nano-drones can accomplish individual intelligence autonomous navigation tasks by leveraging deep learning-based (DL) algorithms – spanning a broad spectrum of complexity and sensorial input [36, 37, 19, 99]. In [36], Kooi and Babuška use a deep reinforcement learning approach with proximal policy optimization for the autonomous landing of a nano-drone on an inclined surface. The designed convolutional neural network (CNN) requires about 4.5 k multiply-accumulate (MAC) operations per forward step, being sufficiently small to allow a single action evaluation in about 2.5 ms on a single-core Cortex-M4 processor, but still providing limited perception capabilities, restricted to landing purposes only. Employing the same Cortex-M4, Neural-Swarm2 [37] exploits a DL-based controller to compensate close-proximity interaction forces that arise in formation flights of nano-drones. With only about 37 kMACs, each nano-drone processes only the relative position and velocity of surrounding UAVs, enabling safe close-proximity flight.

Focusing on higher complexity DL-based workloads, the computational limitations imposed by single-core MCUs can be addressed by leveraging cutting-edge flight controllers targeting artificial intelligence. An example of these new-generation devices is given by the 9-core GWT GAP8[1], a parallel-ultra-low-power (PULP) processor. This processor was previously exploited on UAVs in the PULP-Dronet project [99], leading to a fully-programmable end-to-end visual-based autonomous navigation engine for nano-drones. PULP-Dronet is a single CNN capable of navigating a 27-grams nano-drone in both indoor and outdoor environments by predicting a collision probability, for obstacle avoidance, and a steering angle, to keep the drone within a lane. Similarly, in the PULP-Frontnet project [19] the PULP paradigm is leveraged to successfully run a lightweight CNN (down to 4.8 MMAC and 78 kB) that performs a real-time relative pose estimation of a free-moving human, on a nano-UAV. This prediction is then fed to the nano-drone's controller, enabling precise "human following" capability. Both PULP-Dronet and PULP-Frontnet demonstrate the feasibility of embedding high-level intelligence aboard

[1] https:// greenwaves-technologies.com/gap8_mcu_ai

nano-UAVs. However, all these State-of-the-Art (SoA) solutions focus on vision-based single task intelligence, as the computational/memory burden for multi-tasking perception is still challenging ultra-low-power MCUs.

This Section presents a general methodology for analyzing the size and complexity of deep learning models suffering from *overfitting* and *sparsity*, which we apply, as an example, to the SoA PULP-Dronet CNN. Then, we study the various trade-offs between the number of channels, pruning of inactive neurons, and accuracy, demonstrating the effectiveness of our method by introducing novel squeezed versions of the PULP-Dronet called Tiny-PULP-Dronets. Our CNNs are up to 50× smaller and 8.5× faster than the baseline running on the same PULP GAP8 SoC, with no compromise on the final regression/classification performance. Ultimately, the Tiny-PULP-Dronets, with a minimum model size of only 6.4 kB and a maximum throughput of more than 160 frame/s, enable higher reactivity on the autonomous navigation task, and leave sufficient memory and compute headroom for onboard multi-tasking intelligence even at the nano-sized scale.

5.1.1 Methodology

PULP-Dronet [99] is a ResNet-based [91] CNN made of three residual blocks (ResBlocks), where each one consists of a main branch, performing two 3×3 convolutions, and a parallel by-pass (Byp), performing one 1×1 convolution. This CNN produces a steering angle (regression) and a collision probability (classification) output. Therefore, the model is trained using two different metrics: the mean squared error (MSE) and the binary cross-entropy (BCE). The two metrics are then combined in a single loss function, $Loss = MSE + \beta BCE$, where β is set to 0 for the first 10 epochs, gradually increasing in a logarithmic way to prioritize the regression problem. Finally, the training process exploits a dynamic *negative hard-mining* procedure, gradually narrowing down the loss computation to the k-top samples accounting for the highest error.

PULP-Dronet is deployed in fixed-point arithmetic on a multi-core GAP8 SoC, yielding 41 MMAC operations per frame and 320 kB of weights. The peak memory footprint – also including input and intermediate buffers – is as much as 400 kB, which is close to the total on-chip L2 memory (i.e., 512 kB) and represents a strong limiting factor for our multi-tasking objective.

Overfitting. A strong indicator of a deep learning model's overfitting is a decreasing/constant training loss paired with a validation loss that grows over the epochs. In Figure 5.1, we show both BCE (A) and MSE (B) curves over 100 epochs for both training (64 k images) and validation (7 k images) procedure. In Figure 5.1-A, the *baseline* PULP-Dronet (dotted lines) suffers from overfitting on the BCE, showing a constantly growing validation error, while the MSE (Figure 5.1-B) shows a noisy but almost constant validation error trend. This behavior gives us the intuition

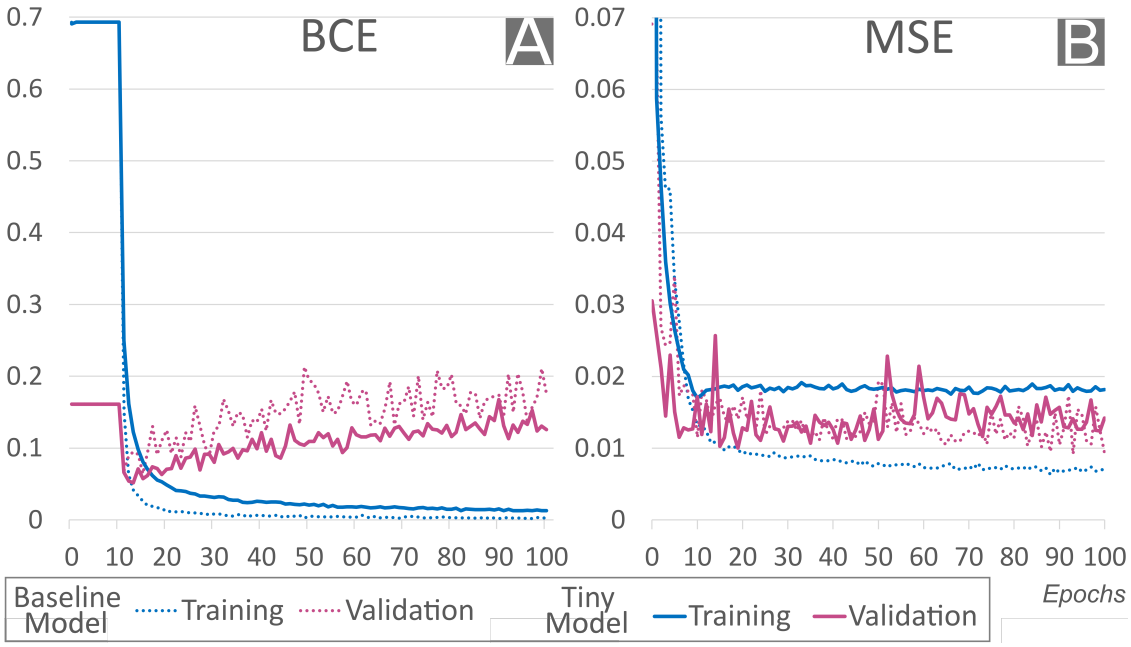


Figure 5.1. Training/validation loss' components – BCE (A) and MSE (B) – of PULP-Dronet, comparing the the baseline model against the tiny one.

that the network topology could be over-parameterized for the given perception problem.

Therefore, we introduce a model exploration to minimize its size while aiming at the same minimum in the validation losses – or even lower. Our approach aims at thinning the network's tensors by applying a γ scaling factor to the number of channels across all layers. We span the γ parameter in the range $[0.125, 0.250, 0.5, 1.0]$, where $\gamma = 1.0$ corresponds to the baseline PULP-Dronet, and we call the smaller variants Tiny-PULP-Dronet CNNs. The smallest Tiny-PULP-Dronet ($\gamma = 0.125$) losses are reported in Figure 5.1 (solid lines), where both the baseline and the tiny model assess to the same minimum validation MSE of 0.01. On the other hand, the tiny model achieves a minimum validation BCE of 0.052, slightly lower than the corresponding value of the baseline (0.064), confirming overfitting of the baseline PULP-Dronet.

Sparsity. Deep learning models' sparsity analysis is a key tool for identifying and selectively pruning parts of the models that are not contributing to the learning process [100]. We define as the *structural activation's sparsity* the percentage of neurons in a convolutional layer that never activate over the entire validation dataset – always equal to 0. This analysis for the PULP-Dronet baseline is reported in Table 5.1, which highlights significant sparsity across the whole network, peaking in the ResBlock3 (92% for the last by-pass). These results suggest: *i)* the network

might suffer of over-parametrization, across all layers; and *ii)* by-pass branches, usually exploited in very deep CNNs to avoid *vanishing gradient* effects [91], are underutilized in this shallow CNN.

Therefore, we analyze the gradients for each network layer after the by-pass removal: the convolution associated with Act1 shows the strongest gradient's attenuation. On this layer, we record on the last 10 epochs (worst case) an average gradient magnitude of 0.055 with a standard deviation of 0.16, supporting the initial intuition of no vanishing gradients on the PULP-Dronet shallow CNN. Furthermore, in Table 5.1, we also provide the same sparsity analysis for the smallest Tiny-PULP-Dronet ($\gamma = 0.125$), with by-pass branches removed. Compared to the baseline model, the tiny one scores a sparsity of almost 0% across all layers, suggesting a more efficient usage of the neurons left after the two squeezing techniques.

Table 5.1. Structural sparsity analysis of PULP-Dronet, comparing the baseline vs. the tiny model.

	Conv	**ResBlock1**			**ResBlock2**			**ResBlock3**		
Topology	Act0	Act1	Act2	Byp1	Act3	Act4	Byp2	Act5	Act6	Byp3
Baseline	28%	13%	6%	6%	25%	10%	11%	49%	60%	92%
Tiny	0.15%	0%	0%	—	0%	0.07%	—	0%	0%	—

5.1.2 Results

Models evaluation

In this section, we present our study on the effect of the proposed methodology when applied to the PULP-Dronet CNN in terms of memory footprint, computational effort, and regression/classification performance. Figure 5.2 presents the comparison between the baseline CNN against its Tiny variants. The reduction of the number of channels through γ saves 237 to 314 kB and 29 to 39 MMAC. Additionally, also the by-pass removal reduces the memory usage of 1 to 12 kB and the operations by 0.1 to 1 MMAC, depending on the specific model. Combining by-pass removal and the scaling of network's channels, the smallest Tiny-PULP-Dronet reduces 50× memory footprint and 27× on MAC operations vs. the baseline, reaching a minimum of 6.4 kB and 1.5 MMAC, respectively.

Focusing on the regression performance for the testing set, the root mean squared error (RMSE) shows a similar trend for both *with* and *without* (w/o) by-pass groups (Figure 5.2). The score slowly improves while reducing the model size from $\gamma = 1$ to $\gamma = 0.25$, which is a common behavior of those models that suffer from overfitting and take advantage of an increased generalization capability, reducing their trainable parameters [100]. Instead, for both groups, the smallest models ($\gamma = 0.125$)

show a lower improvement on the RMSE, suggesting their potential underfitting. Figure 5.2 also reports the Accuracy evaluation for the classification problem. While this metric, on the *with by-pass* group, seems to keep an almost constant performance all over the sizes (∼0.91), it slowly reduces with the size for the *without by-pass* group. Overall, the by-pass removal does not significantly penalize either the RMSE or the Accuracy of the models, allowing to reach a remarkable 0.88 Accuracy for the smallest configuration. More importantly, this pruning is highly desirable for *i*) reducing the memory footprint (∼ 3%) and operations (∼ 1.5%), and *ii*) simplifying the deployment process on MCUs.

To support the second point, we further analyze the peak memory allocation needed for one single-image inference of the CNNs accounting for input/output intermediate buffers and network weights. Looking at the peak memory allocation utilizing a simple *incremental allocator*, which sums up the memory required by each layer, the baseline PULP-Dronet requires 870 kB compared to only 105 kB for the smallest Tiny-PULP-Dronet ($\gamma = 0.125$). On the other hand, by employing a *dynamic allocator* which maximizes data reuse, the baseline PULP-Dronet peaks at 400 kB, while the smallest CNN peaks at 80.1 kB. In this latest case, the removal of by-pass branches simplifies and reduces the memory burden eliminating the need for the simultaneous storage of two intermediate activations along parallel branches, which peaks at 10 kB in the smallest Tiny-PULP-Dronet. Considering the dynamic allocator, we ultimately reduce by 5× the peak memory usage required by the onboard intelligence, a key element to enable multi-tasking aboard constrained devices like the GAP8 SoC.

Power analysis

We evaluate GAP8's execution time and energy consumption when running a single-frame inference with the smallest Tiny-PULP-Dronet model ($\gamma = 0.125$). We use the RocketLogger data logger [92] (64 ksps) to separately plot the power waveforms of the GAP8's main core, the Fabric Controller (FC), and its 8-core parallel cluster (CL). We test two SoC configurations: the so called energy-efficient configuration [99], whose operating point is FC@50 MHz, CL@100 MHz, and VDD@1 V, and the maximum performance settings according to the GAP8's datasheet, which is FC@250 MHz, CL@175 MHz, VDD@1.2 V.

GAP8 takes 1.1 Mcycles to process the Tiny-PULP-Dronet, as shown with the power traces in Figure 5.3, where we highlight the individual execution of each network's layer, for both SoC configurations. The energy-efficient configuration (Figure 5.3-A) shows an average power consumption of 34 mW, 11.3 ms inference time, for a total energy consumption of 0.38 mJ. Moving to the maximum performance configuration (Figure 5.3-B), the inference time gets reduced to 6.3 ms under the same average power, leading to 0.63 mJ per frame. Overall, these results show

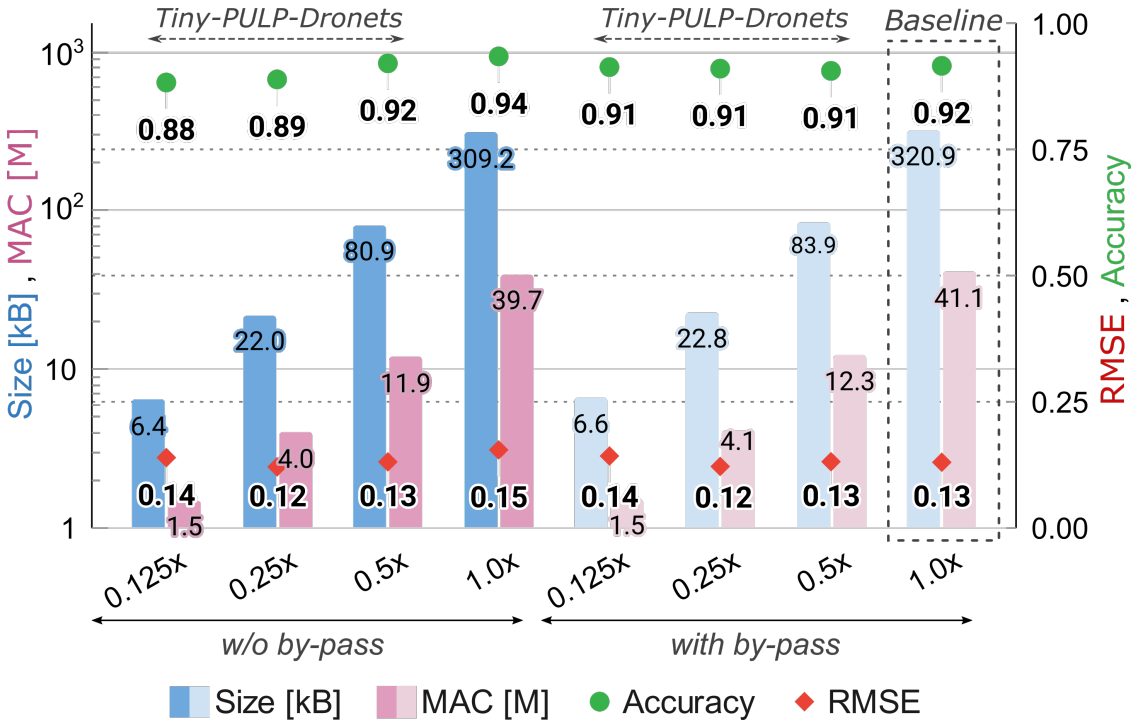


Figure 5.2. Comparing PULP-Dronet vs. its Tiny variants, in terms of size, MAC, Accuracy, and RMSE. We span γ in the $[0.125, 0.250, 0.5, 1.0]$ range.

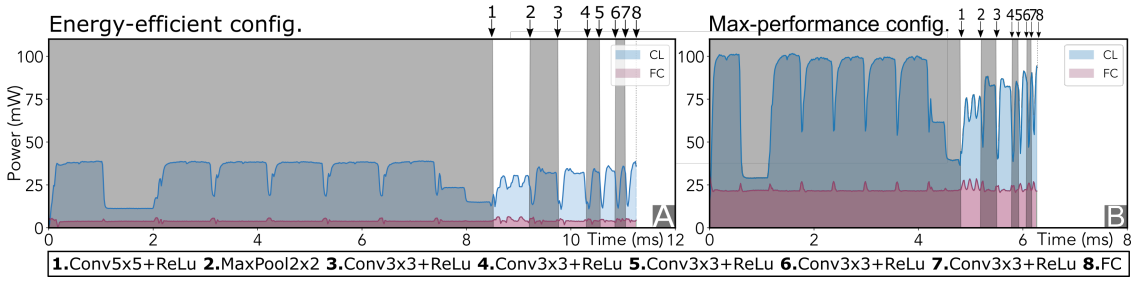


Figure 5.3. GAP8 power waveforms executing the smallest Tiny-PULP-Dronet ($\gamma = 0.125$). (A) most energy-efficient SoC's configuration – FC@50 MHz, CL@100 MHz, VDD@1.0 V – and (B) the maximum performance one – FC@250 MHz, CL@175 MHz, VDD@1.2 V.

an improvement of 8.5× on the network's inference time when compared to the baseline PULP-Dronet, which takes 96 ms and 52 ms to run an inference under the energy-efficient and maximum performance SoC configurations, respectively. Ultimately, we improve the frame-rate of PULP-Dronet from 10 frame/s to 89 frame/s on the energy efficient configuration, and from 19 frame/s to 160 frame/s, on the maximum performance one.

Lastly, we analyze the execution of the first 5×5 convolution, which takes about 75% of the total network's execution time on the smallest Tiny-PULP-Dronet ($\gamma = 0.125$). This layer processes a $200 \times 200 \times 1$ input image and outputs a $100 \times 100 \times 4$ feature map, resulting in 1 MMAC operations, which corresponds to the 70% of the total network's operations, partially explaining its higher execution time over the others. However, alongside the reduction by 27× of the network's MAC operations w.r.t. the baseline, we only witness a latency reduction of 8.5×. This non-linear scaling is due to inevitable non-idealities: *i*) the Height-Width-Channel data layout limits the input feature map data reuse, being proportional to the layer's output channels number, i.e., only 4, *ii*) the marshaling stage required by the convolution for padding and constructing the input's flattened buffer adds up to 45% of the total layer execution. Ultimately, with a peak 8.5× throughput improvement, Tiny-PULP-Dronets enable a faster reactivity of the nano-UAV when autonomously navigating the environment.

5.1.3 Conclusion

The limited payload and computational power of nano-UAVs prevent high-level onboard intelligence, such as multi-tasking execution, which is still out of reach.

This Section presents a general methodology that leverages the CNN's sparsity and overfitting to squeeze both memory footprint and computational effort. By applying our methodology to the SoA PULP-Dronet CNN for autonomous driving, we introduce its Tiny-PULP-Dronet variants. These CNNs show a reduced memory burden (50× smaller) and computational complexity (27× lower) vs. the original model while *i)* preserving the same regression performance, *ii)* having minimal accuracy degradation (6% at most), and ultimately leaving sufficient resources for faster and lighter inference on multi-tasking autonomous nano-drones.

5.2 Distilling Tiny and Fast Deep Neural Networks for Autonomous Navigation on Nano-UAVs

Figure 5.4. Our autonomous nano-UAV navigating an unknown environment.

With the growth of the Internet of Things (IoT), tiny nano-sized unmanned aerial vehicles (UAVs) empowered with onboard artificial intelligence (AI) are gaining importance as ubiquitous IoT nodes: with their sub-10 cm diameter and tens of grams in weight, they can autonomously navigate environments while simultaneously sensing and analyzing their surroundings [20]. Thanks to their compact form factor, nano-UAVs can operate in confined/narrow spaces [1] and safely in the proximity of humans [38] (as shown in Figure 5.4), enabling many use-cases, such as the exploration of harmful environments [4] and rescue missions [101, 102].

A UAV that acts as an intelligent IoT node requires the concurrent execution of multiple tasks in real-time [101], from simple ones (e.g., hovering, state estimation) [43] to complex AI workloads [1, 38]. However, this ideal scenario is challenged by the constrained payload and power limitations of nano-drones, which bound the capabilities of their mission computers to MicroController Units (MCUs) [1]. The State-of-the-Art (SoA) MCU for commercial-off-the-shelf (COTS) nano-drones is the GAP8 System-on-Chip (SoC) [56], a parallel ultra-low-power (PULP) 9-cores processor with a peak throughput of 5.4 GOps/s [68].

In this context, overcoming the computational/memory burden imposed by ultra-low-power MCUs is paramount, yet very challenging [103]. A particularly pivotal requirement for any IoT node moving within an environment is *visual navigation*, i.e., the capability to autonomously navigate an environment based purely on local visual information, avoiding obstacles. The SoA convolutional neural network (CNN) for visual-based autonomous navigation on nano-drones is PULP-Dronet [1], outputting a collision probability and a steering angle suggestion following visual cues in an environment, such as lanes on a floor. This CNN has been demonstrated in indoor and outdoor environments, enabling turns and avoiding collisions with dynamic obstacles. However, these capabilities come at a non-negligible computational cost, which allows for a maximum throughput of 19 FPS on the GAP8 platform [1]; moreover, this CNN shows limited performance to drive the nano-UAV to navigate around static obstacles, limiting its applicability in real-world scenarios.

In this Section, we set a new milestone in the SoA of autonomous visual-based nano-UAV navigation by tackling the avoidance of static obstacles while simultaneously shrinking the CNN memory footprint and computational burden to the minimum. Some of the shortcomings of the PULP-Dronet network, particularly the limitations in obstacle avoidance, depend on the choice to construct the training set by combining datasets from different robotic domains with disjoint training for steering and collision labels. A route to attack this issue is to base the training on data collected directly on a nano-UAV, with the joint information on the presence of an obstacle and the route to avoid it imbued in the training set itself. To do so, we introduce several novel contributions:

- a methodology to collect *unified collision avoidance and steering* information only with onboard resources of the nano-UAV, without dependence on external infrastructure like motion capture systems. The resulting *PULP-Dronet v3* dataset consists of 66 k labeled images, which we release open-source along with our dataset collection framework.

- The end-to-end training and deployment of the first family of CNNs that enable *visual-based static obstacle avoidance and autonomous navigation* suited to nano-drones, *PULP-Dronet v3*.

- an extensive study of minimizing the CNNs' memory footprint and workload for autonomous nano-UAV navigation, resulting in a new family of *Tiny-PULP-Dronet v3* CNNs.

We validate all our networks on our collected dataset and characterize their end-to-end execution time on the GAP8 SoC. Our tiniest network has only 2.9 k parameters and 0.6 M multiply-accumulate (MAC) operations, 168× smaller and 7.3× faster (up to 139 FPS) than the SoA PULP-Dronet v2 [1] when running on the same PULP

GAP8 SoC. This outcome allows us to free up precious computational resources that can be exploited to tackle additional tasks. This network only requires 0.7 mJ for each inference, with an average power consumption of 100 mW when running on GAP8 at its maximum performance.

Furthermore, we deployed and extensively field-tested two networks: a larger one (matching the size of the SoA CNN) and our ultra-tiny distilled CNN. This allowed us to evaluate how the reduced workload affects navigation accuracy. We set up a challenging scenario consisting of a narrow corridor with four static obstacles and a 180° turn. When setting the target speed to 0.5 m/s, both our large and tiny networks exhibit a high success rate in navigating through the corridor, scoring a 80% and 100% success rate, respectively. In the same scenario, the SoA PULP-Dronet v2 [1] always fails. Among our two networks, the larger one shows better robustness at higher speeds, succeeding 60% with a target speed of 1 m/s, whereas the tiny one fails. We foster the research on autonomous nano-drone navigation by releasing all our results as open-source: our new dataset, dataset collector framework, CNN weights, and code.

5.3 Nano-UAV robotic platform

5.3.1 Robotic platform and the GAP8 SoC

The robotics platform employed in this section utilizes the Bitcraze Crazyflie 2.1 quadrotor, as introduced in the background chapter (Chapter 3). The setup includes the main flight controller PCB equipped with an STM32 MCU and two additional pluggable PCBs: the Flow deck and the AI-deck. The PULP-Dronet v2 CNN introduced in this chapter is executed on the GAP8 SoC, introduced in Chapter 3. This chapter examines a specific configuration where the Wi-Fi and radio modules of the UAV are inactive during autonomous navigation tasks, relying solely on the visual data from the front-facing camera. This setup ensures that the drone operates completely autonomously, with all navigation intelligence residing on the nano-drone, independent of external communication or computational support. However, radio and Wi-Fi connections are utilized exclusively for dataset collection (refer to Section 5.4.2). Additionally, we use the multi-ranger deck to measure distances to front-facing obstacles for data acquisition purposes only.

5.3.2 Automatic Deployment Tools

The development of CNNs for an MCU-class processing device, such as the GAP8 on our nano-drone, is a multi-objective optimization problem. In our case, it must take into account: *i*) memory limitations (L2 512 kB and L1 64 kB), *ii*) the hardware limitations (i.e., no floating point unit), power envelope (∼100 mW), and

throughput (i.e., a flying drone needs to react in real-time). As for PULP-Dronet v2 [1], we use an automated flow that works in two steps: first, we quantize the neural network, and then we perform a hardware-aware deployment of the quantized model.

For quantization, we take `float32` pre-trained CNN models, and we apply post-training quantization using NEMO [104], a deep neural network (DNN) quantization tool that performs uniform asymmetric static layer-wise quantization. We apply fixed-point 8-bit (`int8`) post-training quantization to the weights and activations of our networks. By doing so, we enable optimized 8-bit fixed-point arithmetic on GAP8, i.e., packed-SIMD instructions [64]. Moreover, the conversion from `float32` data type and the quantized `int8` reduces by 4× the memory footprint of the CNNs models.

For hardware-aware deployment, we use DORY[59], a state-of-the-art code generation tool for quantized DNNs. It is tuned for deployment on memory-constrained embedded devices with multiple levels of memory hierarchy with the goal of optimizing performance. To fit the data in the available memory resources, large operations need to be split into smaller pieces called tiles. DORY models the tile size optimization problem as a constraints program with the memory size as the main constraint and hardware-aware heuristics to guide the ILP solver towards the best-performing solutions. While tiling makes the operations fit into our desired memory, it still requires data marshalling between the memory levels to process the whole layer. To hide this data movement cost, DORY overlaps it with computation by employing multi-buffering and software pipelining. For efficient operation computation, calls to the PULP-NN kernels get generated. PULP-NN[68] is an open-source library of hand-optimized kernels for quantized neural networks executing on the PULP cluster.

5.3.3 Baseline PULP-Dronet CNN

PULP-Dronet [1] is an end-to-end vision-based CNN for autonomous navigation aboard nano-drones, deployed on a nano-drone for the first time in [18] and suited for deployment on the AI-Deck. We employ PULP-Dronet as a baseline model for our work. This shallow NN is based on three residual blocks (ResBlocks) [91], where each one consists of a main branch, performing two 3×3 convolutions, and a parallel by-pass, performing one 1×1 convolution. The number of output channels is 32, 64, and 128 for the three blocks, respectively. For each block, the last convolution on both branches of applies a stride factor of 2 to half the feature map size. Each convolution is followed by a Batch Normalization layer and by a ReLU6 activation function.

The CNN processes a grayscale image of size 200×200. This image is the bottom-center crop extracted from the QVGA image captured by the AI-Deck's Himax

Figure 5.5. A) our dataset collection overview, B) our dataset collector GUI, and C) a sample sequence from our collected dataset.

camera. The final layer produces two distinct outputs: a collision probability (classification problem) and a steering angle (regression problem). Consequently, the model is trained using two distinct metrics: the mean squared error (MSE) and the binary cross-entropy (BCE). These metrics are combined into a unified loss function, $Loss = MSE + \beta BCE$. β is set to 0 for the first 10 epochs, gradually increasing in a logarithmic way to prioritize the regression problem. The training process employs a dynamic *negative hard-mining* approach, gradually focusing the loss computation on the k-top samples that exhibit the highest error. PULP-Dronet is deployed in fixed-point 8-bit arithmetic on the multi-core GAP8 SoC, yielding 41 MMAC operations per frame and 320 kB of weights.

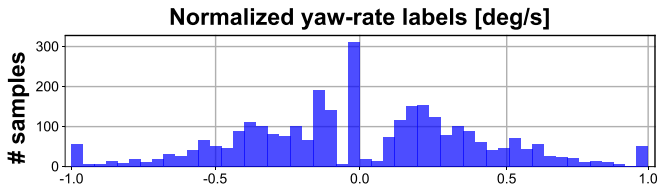
The original dataset used for training, validation, and testing integrates three disjoint sets of images: Udacity (39.1 k) consisting of high-resolution images labeled solely with steering angles, Bicycle (32.2 k) containing high-resolution images labeled exclusively with collision probabilities, and Himax (1.3 k) comprising low-resolution images captured from the same camera aboard our target nano-drone, labeled only with collision probabilities. The combination of Udacity, Bicycle sets, and Himax forms the *Original+Himax* dataset used to train PULP-DroNet V2.

5.4 Methodology

5.4.1 The dataset collector tool

We developed a software tool for dataset collection, outlined in Figure 5.5-A. This tool enables a pilot to manually control the nano-drone and simultaneously log: *i*) any data from the STM32 flight controller (e.g., the drone's state estimation) and sensors attached to it (e.g., additional expansions decks of the Crazyflie2.1[2], such as ranging sensors), *ii*) images captured by the AI-deck's front-facing camera. All data is recorded on a PC base station, which independently receives the two

[2] `https://store.bitcraze.io/collections/decks`



Trivial Predictor	Collision (Acc)	Steering (RMSE)
Never Collision	**0.58**	**N/A**
Always Collision	**0.42**	**N/A**
Always straight	**N/A**	**0.41**

Figure 5.6. Distribution of the steering labels in our testing dataset and classification/regression performances of three trivial predictors.

data streams: the STM32 packets, transmitted via radio, and AI-deck images, transmitted via Wi-Fi.

To ensure the data from these two separated streams can be accurately matched in post-processing, we *i*) periodically synchronize both the STM32 and GAP8 MCUs with a global clock, generated from a GPIO of GAP8, and *ii*) send each chuck of data associated with a timestamp that is globally synchronized across our robotic platform. This synchronization methodology minimizes matching errors by ensuring each packet is timestamped at its source, thus avoiding network-related errors and leaving only minimal discrepancies caused by the MCUs' oscillator drifts.

Our dataset collector tool consists of: *i*) code for the STM32 and GAP8 SoC (Figure 5.5-A), which enables sending the data streams while keeping the global clock synchronized, and *ii*) a graphical user interface (GUI) (Figure 5.5-B) to plot the data streamed from the flight controller and visualize images collected from the AI-deck.

5.4.2 Our dataset for nano-drone autonomous navigation

We introduce a new dataset for visual-based autonomous nano-drone navigation. We move on from the limitations of the past PULP-Dronet dataset [18], which was created by assembling different sets of data, each one having images labeled either for the steering or the collision avoidance task, as explained in 2.2.3, ultimately penalizing the static obstacle avoidance [1]. To tackle this limitation, we collect a new unified dataset from scratch.

With the dataset collector tool introduced in Section 5.4.1, we collected a dataset of 66 k images for nano-drones' autonomous navigation, for a total of ∼600 MB of data. We used the Bitcraze Crazyflie 2.1, described in Section 3.1. A human

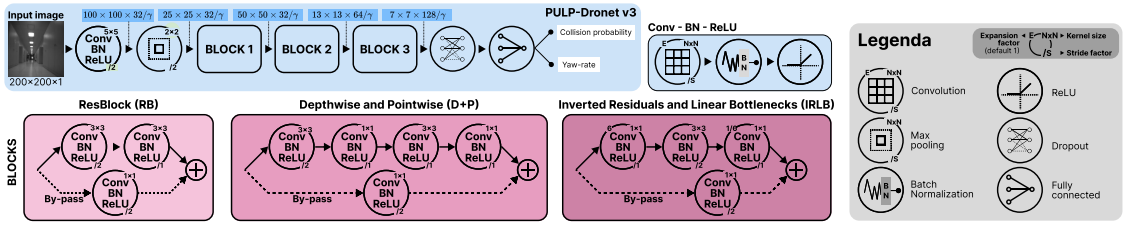


Figure 5.7. Our network architecture exploration includes: *i*) three block types – RB, D+P, IRLB; *ii*) an optional bypass connection (dashed line); *iii*) variations on the number of channels based on γ.

pilot manually flew the drone, collecting *i*) images from the grayscale QVGA Himax camera sensor of the AI-deck, *ii*) the gamepad's yaw-rate, normalized in the $[-1;+1]$ range, inputted from the human pilot, *iii*) the drone's estimated state, and *iv*) the frontal obstacle's distance with the ToF sensor.

After the data collection, we labeled all the images with a binary collision label whenever an obstacle was in the line of sight and closer than 2 m. We recorded 301 sequences in 20 different environments. Each sequence of data is labeled with high-level characteristics: scenario (i.e., indoor or outdoor), path type (i.e., presence or absence of turns), obstacle types (e.g., pedestrians, chairs), flight height (i.e., 0.5 m, 1 m, 1.5 m), light conditions (dark, normal, bright), acquisition date, and a location name identifier.

For training and validating our CNNs, we post-processed the datasets as follows. We used 70%, 10%, and 20% of the images as training, validation, and testing sets, respectively. We augmented the training images by applying random flipping, brightness augmentation, vignetting, and blur. The resulting training dataset has ∼124 k images, split as follows: 109 k, 5 k, 10 k images for training, validation, and testing, respectively. To address the labels' bias towards the center of the $[-1;+1]$ yaw-rate range in our testing dataset – specifically, the over-representation of images associated with a yaw-rate of 0, indicating no input from the human pilot and thus the drone flying straight – we balanced the dataset by selectively removing a portion of images that had a yaw-rate of 0. The final distribution of our balanced dataset is represented in Figure 5.6. In the same Figure, we also report the RMSE and Accuracy for three trivial predictors, i.e., a predictor that either always predicts collision to 1 or 0, and a predictor that always predicts a yaw-rate of zero (i.e., go straight). These values can be later used in Section 5.5 as a baseline to assess the RMSE and Accuracy performance of our trained networks.

5.4.3 Network architecture design

The original PULP-Dronet architecture [1] exploits three ResBlocks (RB) [91]. In other vision tasks, the RB architecture has been largely replaced by other kinds of network topologies, which provide similar accuracy characteristics but with lower workload and memory footprint [57]. Therefore, we design the new PULP-Dronet v3 architecture by exploring several modifications to its baseline topology.

Our modifications, detailed in Figure 5.7, involve substituting the three RB blocks of PULP-Dronet v2, removing parallel by-pass branches, and varying the number of channels in the intermediate feature maps. To optimize autonomous navigation and enable more complex onboard functionality, we studied several modifications to this setup. First, we replace these blocks with two other blocks we designed, taking inspiration from the well-known Mobilenet family of CNNs [105, 58]. These CNNs have been shown to be both accurate for visual AI tasks and suitable for efficient MCU deployment [106, 57]. For each block we substitute, we keep the same output feature map dimensions (width and height) of PULP-Dronet v2.

The first new block we design is based on the Mobilenet v1 [105]. It uses separable depthwise and pointwise (D+P) convolutional layers instead of traditional ones. Such layers factorize a standard convolution into a sequence of K×K *depthwise* convolution and a 1×1 convolution called *pointwise* convolution, where K is the kernel size. We define our D+P block as consisting of two branches: the main branch performs two 3×3 D+P convolutions, while the parallel branch executes a standard 1×1 convolution. We apply convolutional strides different from 1 only to the last convolution in both branches. Each convolution (depthwise, pointwise, and standard) is subsequently followed by a batch normalization (BN) layer and a rectified linear unit (ReLU). This Conv-BN-ReLU pattern simplifies both quantization and deployment explained in Section 5.5.4: the tensor outputted by the Conv operation demands a finer grain representation (utilizing 32 bit) compared to the inputs and weights. Therefore, the BN is used to accumulate a 32 bit representation, while the ReLU reduces it back immediately to 8 bit.

The second block we define is inspired by Mobilenet v2 [58], using inverted residuals and linear bottlenecks (IRLB) layers. This block comprises a 1×1 convolution, which expands the number of channels by an expansion factor, succeeded by a D+P convolution, and a 1×1 projection convolution that inverts the expansion, reducing the number of output channels. Stride factors different from 1 are applied in the D+P convolution. We choose an expansion ratio equal to 6 as in [58]. Additionally, we add a by-pass branch performing a 1×1 convolution, ensuring the same output size as the main IRLB branch. expanding to a higher-dimensional feature space

We explore additional architecture variations by considering the removal of parallel by-pass branches from each block type. These branches primarily serve to mitigate

vanishing gradient effects in deep CNNs [91]. However, recent studies [103] have highlighted their inefficacy in shallow CNN models, such as the seven-layer PULP-Dronet.

Lastly, we explore different sizing factors to investigate the accuracy, memory footprint, and computational cost trade-offs, As in [103, 105], we thin the network's tensors by applying a γ dividing factor to the number of channels across all convolutional layers. We span the γ parameter in the range $[1, 2, 4, 8]$, where $\gamma = 1$ corresponds to the baseline size for PULP-Dronet[1].

5.5 Experimental Results

5.5.1 Datasets evaluation

In Table 5.2, we assess the impact of training and testing on our new dataset (**v3**) versus the PULP-Dronet v2 dataset introduced in [18] (**v2**); each dataset is split into different training and testing sets. We perform this assessment, keeping the original PULP-Dronet CNN architecture in both cases. When the CNN is trained on v2 and tested on v2, we achieve performance consistent with the results reported in [1]. When training on v3 and testing on v3, both performances decrease compared to the previous case due to the higher complexity of the v3 testing set, such as a more uniform distribution of steering labels and a richer, and therefore, more challenging, dataset. We also report the cross-validation of training on v2 and testing on v3, and training on v3 and testing on v2. In both cases, performances drop compared to training and testing on the same dataset version, but the v2-trained network (tested on v3) has a higher drop on both RMSE and Accuracy than the v3-trained (tested on v2), suggesting better generalization capabilities for the proposed dataset.

Table 5.2. Cross-validation between PULP-Dronet v2 dataset and ours (v3).

Training \ **Testing**	**v2** (6.9 k images)		**v3** (10 k images)	
	RMSE	**Accuracy**	**RMSE**	**Accuracy**
v2 (64 k images)	0.118	90%	0.556	54%
v3 (109 k images)	0.236	67%	0.350	83%

5.5.2 CNN architectures exploration

To evaluate the PULP-Dronet v3 family of networks proposed in Section 5.4.3, we start by analyzing different block types (RB, D+P, and IRLB) and the effect of

removing the by-pass branches. In Table 5.3, we assess the networks in terms of parameter count, MAC operations, and key performance metrics – the Accuracy for the classification problem (the higher, the better) and the RMSE for the regression one (the lower, the better).

Removing the by-pass branches negligibly affects the CNN performance metrics on all networks, i.e., at most +0.005 RMSE and +1% in accuracy with IRLB CNN. IOn the other side, by-pass removal reduces network size by $3-11\%$ and number of MAC by $3-15\%$, depending on the architecture. Focusing on the three CNNs without by-pass, the RB variant achieved the lowest RMSE of 0.339, whereas D+P and IRLB models score 0.350 and 0.369, respectively. Instead, the D+P network achieves the highest performance of 84% on the accuracy score. The D+P CNN is also the faster CNN, requiring only 12 MMAC per inference, and the smaller in memory usage, being 6.2 × smaller than RB and 2.5 × smaller than IRLB. Ultimately, we select the D+P model without by-pass branches for its efficiency and minimal performance drop.

Table 5.3. CNN architectures analysis varying computational blocks and residual by-pass. The first row shows the SoA baseline [1].

Blocks	**by-pass**	γ	**Type**	**RMSE**	**Acc**	**MAC**	**Param**	**Size [B]**
RB	yes	/1	fp32	0.339	83%	41M	320k	1.3M
RB	no	/1	fp32	0.339	83%	40M	309k	1.2M
D+P	yes	/1	fp32	0.352	83%	14M	63k	252k
D+P	**no**	**/1**	**fp32**	**0.350**	**84%**	**12M**	**51k**	**204k**
IRLB	yes	/1	fp32	0.369	82%	43M	140k	560k
IRLB	no	/1	fp32	0.364	83%	41M	128k	513k

5.5.3 CNNs size analysis

In this section, we analyze how the number of channels of our CNNs affects their memory requirements, number of operations, and regression/classification performances. Starting from the CNN architecture D+P without by-passes, we span the γ parameter (see Section 5.4.3) in the range $[1, 2, 4, 8]$, where $\gamma = 1$ corresponds to the baseline size for PULP-Dronet[1]. Table 5.4 shows that using a dividing factor $\gamma = 2$ does not affect the classification accuracy of the network when compared to $\gamma = 1$, both scoring 84%. When using smaller networks, the classification accuracy drops by 3% with $\gamma = 4$ and by 7% with $\gamma = 8$.

On the regression task, the RMSE gradually increases as the networks become smaller. For $\gamma = 1$, the RMSE stands at 0.350 , but increase to 0.367 , 0.373 , 0.379 for $\gamma = 2$, $\gamma = 4$, $\gamma = 8$, respectively. On the other hand, the γ factor greatly

Table 5.4. Accuracy, RMSE, MACs, and memory footprint of our architectures by varying γ.

Blocks	**by-pass**	γ	**Type**	**RMSE**	**Acc**	**MAC**	**Param**	**Size [B]**
D+P	no	/1	fp32	0.350	84%	12M	51k	204k
D+P	no	/2	fp32	0.367	84%	5.2M	17k	69k
D+P	no	/4	fp32	0.373	81%	2.4M	6.6k	26k
D+P	no	/8	fp32	0.379	78%	1.1M	2.9k	12k

Table 5.5. Accuracy, RMSE, and memory footprint of our quantized networks.

Blocks	**by-pass**	γ	**Type**	**RMSE**	**Acc**	**MAC**	**Param**	**Size [B]**
D+P	no	/1	int8	0.361	84%	12M	51k	51k
D+P	no	/2	int8	0.373	82%	5.2M	17k	17k
D+P	no	/4	int8	0.378	81%	2.4M	6.6k	6.6k
D+P	no	/8	int8	0.388	78%	1.1M	2.9k	2.9k

impacts the number of the network's parameters. The biggest network ($\gamma = 1$) has 204 k parameters, the network with ($\gamma = 2$) has 3× less parameters (69 k parameters). Increasing the γ to 4 leads to a network with 26 k parameters, and finally the smallest network ($\gamma = 8$), which we call Tiny-PULP-Dronet v3, leads to only 1.9 k parameters.

5.5.4 Quantization and deployment

IIn this section, we progress with quantizing and deploying the four D+P models without by-passes, introduced in Section 5.5.3, to evaluate their trade-off between accuracy/RMSE and onboard execution efficiency. We apply 8-bit post-training quantization, as detailed in Section 5.5.4. Table 5.5 outlines the quantized models' regression/classification, compared to non-quantized models in Table 5.4. Quantization introduces only 2% reduction in accuracy on the network employing $\gamma = 2$, while other CNNs keep the same score of the 32-bit precision. Instead, the RSME is more sensitive to the new int8 data type, with the error increasing of 0.011, 0.006, 0.005, and 0.009, for γ = 1,2,4,8, respectively. Compared to the original float32 models (Table 5.4), we reduce by 4× the memory footprint of each CNN.

Then, we deploy these four quantized networks on the GAP8 SoC to analyze their on-device performances. Table 5.6 shows their inference rate (frame/s) when GAP8 is running at its *max performance (mp)* configuration, i.e., FC@250 MHz,

Table 5.6. Networks' throughput when deployed to GAP8 at its maximum performance configuration. The energy per inference is reported for two configurations (E_{ee}, E_{mp}).

Blocks	**by-pass**	γ	**Cycles**	$\frac{MAC}{Cycle}$	**frame/s**	E_{ee} **[mJ]**	E_{mp} **[mJ]**
D+P	no	/1	5.1M	2.4	34	2.1	3.0
D+P	no	/2	2.9M	1.8	61	1.1	1.7
D+P	no	/4	1.7M	1.4	101	0.6	1.0
D+P	no	/8	1.3M	0.9	139	0.4	0.7

CL@175 MHz, and VDD@1.2 V . Our largest CNN model ($\gamma = 1$) achieves a throughput of 34 frame/s, which is 1.8× higher than the SoA PULP-Dronet v2, peaking at 19 frame/s, despite using the number of channels across the network; this improvement derives from our architecture modifications. Other configurations of γ result in 61 frame/s with 14 kB, 101 frame/s with 4.7 kB, and 139 frame/s with 1.9 kB for $\gamma = [2, 4, 8]$, respectively. Our smallest model ($\gamma = 8$), called Tiny-PULP-Dronet v3, improves the throughput by 7.3× and reduces the memory footprint by 168× compared to PULP-Dronet v2.

Last, we measure the power consumption of all CNNs under SoC configurations: *i*) the max performance setting, and the energy efficient (ee) one, which operates at FC@50 MHz, CL@100 MHz, and VDD@1.0 V. In the ee configuration, the networks with $\gamma \in$ 1,2 consume 38 mW, while the networks with $\gamma \in$ 4,8 show an average power consumption of 34 mW. On the other hand, in the maximum performance configuration, all four networks show an average power consumption of 100 mW. As shown in Table 5.6, the energy for one-frame inference with the ee configuration (E_{ee}) is 2.1 mJ, 1.1 mJ, 0.6 mJ, 0.4 mJ for the γ = 1,2,4,8, respectively, while for the mp configuration (E_{mp}) is always $\sim$ 1.5× higher.

5.6 In-field testing

We evaluate the navigation capabilities of our PULP-Dronet v3 and Tiny-PULP-Dronet v3 CNNs with in-field experiments. We record all experiments in a flying room equipped with a Qualisys motion capture system (24 cameras) with mm-precise tracking of our drone. We track the position and pose of our drone @100 Hz to analyze the drone's flight in post-processing.

We investigate if the new dataset we collected, having unified labels for *yaw-rate* and *collision probability*, improves the navigation capabilities with respect to the SoA PULP-Dronet v2, which was trained with disjoint *steering* and *collision probability*

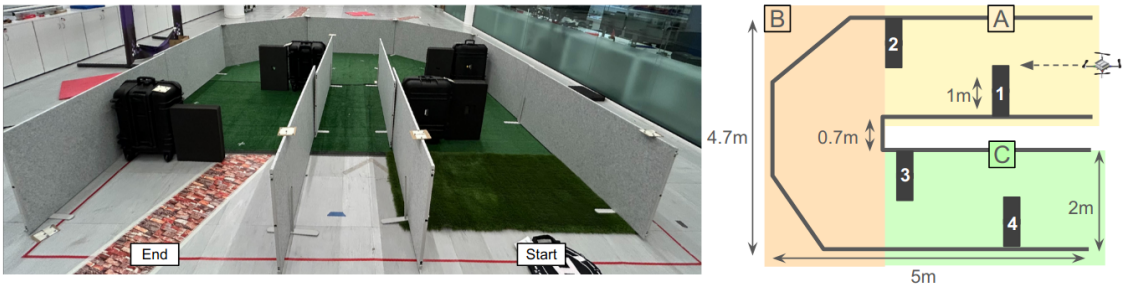


Figure 5.8. Our u-shaped path for the in-field experiments.

labels, and therefore struggles in avoiding static obstacles, as described in [1].

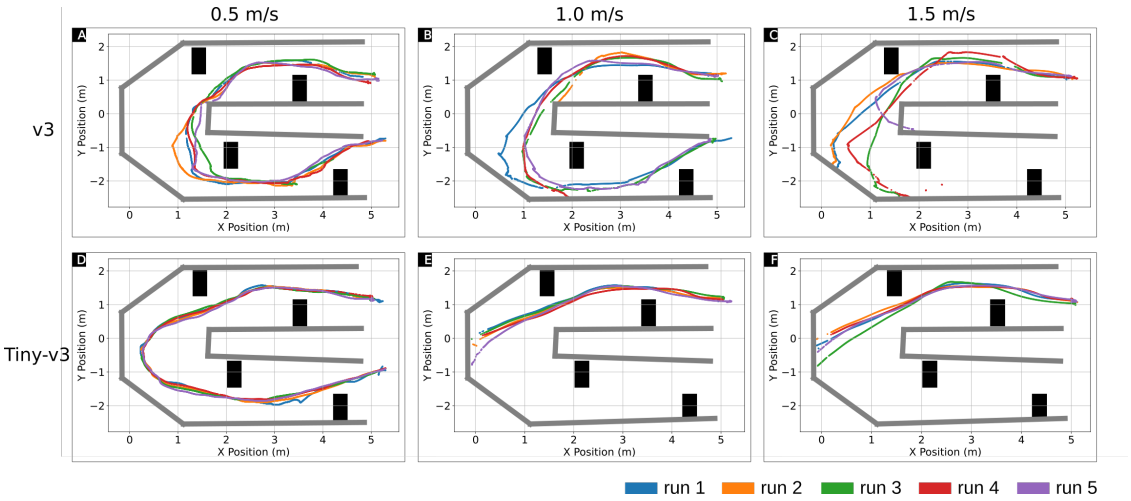


Figure 5.9. In-field experiments trajectories of PULP-Dronet v3 (A-B-C) and Tiny-PULP-Dronet v3 (D-E-F) for three target speeds v_{target} = [0.5,1.0,1.5]m/s, tested with static obstacles only (represented as black rectangles).

We set up a challenging navigation scenario: a U-shaped corridor, represented in Figure 5.8, which we divide into three segments. Segments A and C feature straight paths, each obstructed by two obstacles, whereas segment B encompasses a 180-degree turn. The obstacles have a width of 1 m, and they are attached on opposite sides of the 2 m wide corridor, blocking straight pathways. We designed two experiments: *i*) one where all obstacles are static, and *ii*) one where obstacle 2 is dynamic. In the dynamic scenario, obstacle 2 is initially absent from the scene. It appears in the center of the lane when the drone passes obstacle 1, leaving 1.5 m of braking space, and it is removed 5 s after. These two scenarios challenge the nano-UAV navigation capabilities on multiple skills: *i*) static and dynamic obstacle avoidance, *ii*) navigation through a narrow environment (a corridor), and

Table 5.7. Success rate of our closed-loop system in the u-shaped path with static obstacles only. We report the success rate for the A-B-C section of the path, and the v_{avg} over the complete path for PULP-Dronet v2, PULP-Dronet v3, or Tiny-PULP-Dronet v3.

$\mathbf{v_{target}}$	**PULP-Dronet v2 [1]**				**PULP-Dronet v3 (ours)**				**Tiny-PULP-Dronet v3 (ours)**			
	success rate				success rate				success rate			
	(A)	(B)	(C)	v_{avg} [m/s]	(A)	(B)	(C)	v_{avg} [m/s]	(A)	(B)	(C)	v_{avg} [m/s]
0.5	0/5	0/5	0/5	—	5/5	5/5	4/5	0.39	5/5	5/5	5/5	0.45
1	0/5	0/5	0/5	—	5/5	4/5	3/5	0.24	5/5	0/5	0/5	—
1.5	0/5	0/5	0/5	—	5/5	0/5	0/5	0.96	5/5	0/5	0/5	—

iii) a sharp 180° turn. The corridor environment we use in our tests is *never-seen-before* for both PULP-Dronet v2 and PULP-Dronet v3, not being part of the training set of either network.

We test our closed-loop system at three distinct target speeds $v_{target} = [0.5\,,\,1.0\,,\,1.5\,]$ m/s. In every test, we set 0.5 m as the target height. We conduct five experiments for each combination of CNN and v_{target} for statistical relevance, and we compare with the SoA PULP-Dronet v2 [1]. To keep comparability with the SoA, we always use the same drone's control state machine described in [1]. The videos of the experiments are accessible at[3].

5.6.1 U-shaped path with static obstacles

We conduct the first set of experiments with all static obstacles. Table 5.7 outlines the success rate for each CNN tested. PULP-Dronet v2 never succeeds at any speed point to pass section A of the path. Together, the obstacles obstruct the entire corridor's width, resulting in a consistently high collision probability, while the network's steering output keeps the drone in the center of the lane defined by the surrounding walls. As a result, the drone either does not move forward, due to the CNN's high collision probability, or it slowly drifts against obstacle 1, ultimately crashing. This outcome aligns with a similar scenario described in [1], where PULP-Dronet v2 encountered a 100% failure rate in tackling a narrow tunnel scenario with static obstacles.

Moving to our CNNs, we plot the trajectories of PULP-Dronet v3 in Figure 5.9-A-B-C, and the trajectories of Tiny-PULP-Dronet v3 in Figure 5.9-D-E-F. We start analyzing the results with a target speed of 0.5 m/s. PULP-Dronet v3 and Tiny-PULP-Dronet v3 are able to fly through the whole U-shaped path with a single failure and no failures, respectively. The only time that PULP-Dronet v3 fails, it crashes against obstacle 4, still successfully completing 66% of the whole path. Analyzing the @1 m/s target speed configuration, PULP-Dronet v3 completes the full corridor 3 times while crashing one time in the B section and once in the C section. On the other hand, Tiny-PULP-Dronet v3 reliably succeeds 5/5 times only in section A of the path, always crashing during the B section. This result is explained by the fact that the RMSE performance of Tiny-PULP-Dronet v3 is lower with respect to PULP-Dronet v3 as described in Section 5.5.4. Analyzing the @1.5 m/s target speed configuration, none of the networks tested succeeded in completing the path, outlining the speed upper limit of our closed-loop systems in this scenario. These results mark an improvement with respect to PULP-Dronet v2, showing that *i*) the dataset we collected with joint labels successfully trains networks that can tackle both obstacle avoidance and steering tasks together, and

[3] `https://www.youtube.com/c/PULPPlatform`

ii) our Tiny-PULP-Dronet v3, with only 2.8 k parameters, can enable static obstacle avoidance while being 168× smaller than the SoA model.

5.6.2 U-shaped path with static and dynamic obstacles

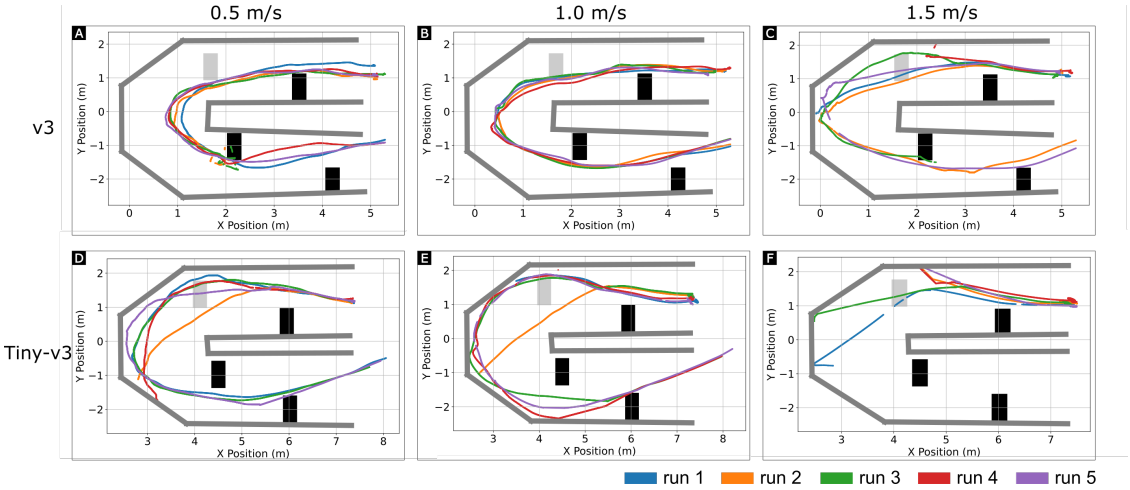


Figure 5.10. In-field experiments trajectories of PULP-Dronet v3 (A-B-C) and Tiny-PULP-Dronet v3 (D-E-F) for three target speeds v_{target} = [0.5,1.0,1.5]m/s, tested with three static obstacles (represented as black rectangles) and a dynamic one (represented as a light grey rectangle).

We conduct the second set of experiments stressing dynamic obstacle avoidance. We use the same setup as Section 5.6.1, but now obstacle 2 appears in the center of the corridor after the drone passes obstacle 1. Table 5.8 outlines the success rate of the three CNNs tested. Starting from the SoA CNN, PULP-Dronet v2 never succeeds in navigating any of the three segments (A,B,C) of our path for the same reason described in Section 5.6.1, getting stuck in front of obstacle 1 and eventually crashing into it.

Moving to our CNNs, we plot the trajectories of PULP-Dronet v3 in Figure 5.10-A-B-C, and the trajectories of Tiny-PULP-Dronet v3 in Figure 5.10-D-E-F. PULP-Dronet v3 shows the highest success rate among all the CNNs tested. It completes the whole path with a success rate of 60%, 100%, 40% when flying at [0.5 , 1.0 , 1.5] m/s, respectively. Remarkably, PULP-Dronet v3 only crashed once against the dynamic obstacle while flying at the highest speed (1.5 m/s), always succeeding in passing the A segment in all other cases. On the other hand, the Tiny-PULP-Dronet v3 completes the whole corridor with a 60% success rate when flying at both 0.5 m/s and 1 m/s. However, it always fails at a higher speed of 1.5 m/s, either crashing on the right wall of Section A or failing to complete the turn in Section B. Nevertheless, Tiny-PULP-Dronet avoided the collision against the dynamic obstacle of section A

Table 5.8. Success rate of our closed-loop system in the u-shaped path with three static obstacles and a dynamic one. We report the success rate for the A-B-C section of the path, and the v_{avg} over the complete path for PULP-Dronet v2, PULP-Dronet v3, or Tiny-PULP-Dronet v3.

$\mathbf{v_{target}}$	**PULP-Dronet v2 [1]**				**PULP-Dronet v3 (ours)**				**Tiny-PULP-Dronet v3 (ours)**			
	success rate				success rate				success rate			
	(A)	(B)	(C)	v_{avg} [m/s]	(A)	(B)	(C)	v_{avg} [m/s]	(A)	(B)	(C)	v_{avg} [m/s]
0.5	0/5	0/5	0/5	—	5/5	5/5	3/5	0.57	5/5	5/5	3/5	0.44
1	0/5	0/5	0/5	—	5/5	5/5	5/5	0.98	5/5	5/5	3/5	0.82
1.5	0/5	0/5	0/5	—	4/5	3/5	2/5	1.33	2/5	0/5	0/5	—

two times when flying at the highest speed.

In conclusion, our experiments showcase the dynamic obstacle avoidance function of both PULP-Dronet v3 and Tiny-PULP-Dronet v3. They consistently avoided the section A obstacle at a speed of 1 m/s. At higher speeds of 1.5 m/s, their success rates were 80% and 40%, respectively.

5.7 Conclusion

Nano-sized UAVs are ideal candidates as ubiquitous flying IoT nodes. In this study, we distill a novel family of networks for autonomous navigation on nano-drones, i.e., the Tiny-PULP-Dronet v3 CNNs. Compared to the SoA, our models reduce memory footprint by up to 168× (down to 2.9 kB) and achieve an inference rate of up to 139 frame/s. We create a new open-source 66 k image dataset for autonomous nano-UAV navigation. We compare with in-field tests both SoA and our networks on a commercially available nano-UAV. Our tiny networks succeed in navigating a challenging path with static and dynamic obstacles and a 180° turn at speeds of up to 1 m/s, whereas the SoA PULP-Dronet consistently fails despite having 168× more parameters.

Chapter 6

AI multi-tasking on nano-UAVs

In this Chapter, we take another step forward in the nano-UAVs SoA toward achieving a higher level of intelligence. As biological entities pursue multi-objective missions in complete autonomy, we aim to enable these nano-UAVs to execute multiple AI algorithms fully onboard. We tackle this complex problem in two steps. First, we tackle a new intelligence task previously unaddressed by autonomous nano drones: object detection. We develop multiple CNNs, trained to recognize two classes of objects, having different trade-offs between accuracy and throughput. The largest and most accurate model scores a mean average precision (mAP) of 50% on an in-field collected dataset while running at 1.6 frame/s on the nano-drone MCU with a power envelope of only 134 mW. With this CNN, we build a nano-drone system that aims at exploring a room while detecting all the objects of interest in it: we combine state-machine-based exploration policies, relying on ToF ranging sensors for collision avoidance, and the vision-based CNN for object detection. The effectiveness of this system is further validated in-field, where it demonstrates an average detection rate of 90% on six target objects in a never-seen-before environment.

Second, building on the progress made in Chapter 5, we enable the execution of multiple AI algorithms on a nano drone: we merge the visual-based autonomous navigation CNN developed in Chapter 5 and the object detection CNN developed in this chapter. This integration allows for the sequential execution of visual-based obstacle avoidance and object detection tasks on autonomous nano-UAVs, running in real-time at 1.6 frames/s. Ultimately, this achievement marks the SoA by presenting the first fully autonomous nano-drone tackling a multi-objective mission, which consists of exploring, preventing collision, and detecting objects in real-time while relying only on onboard sensory and computational resources.

The rapid evolution of the Internet of Things (IoT) fuels the advent of flexible edge nodes powered by artificial intelligence (AI). In this context, AI-based nano-sized

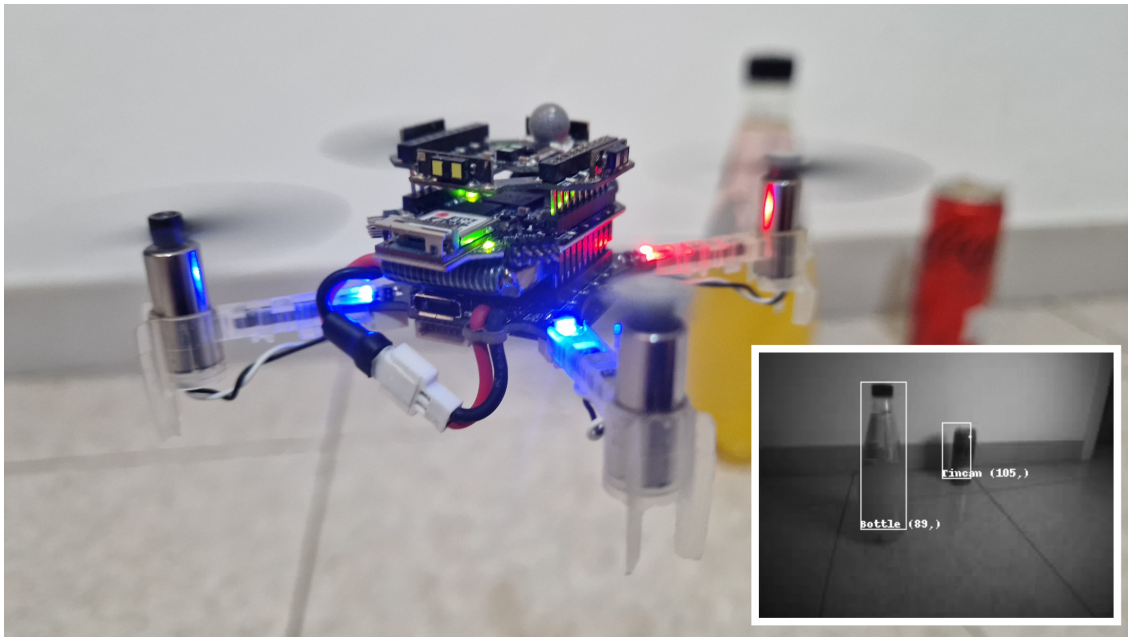


Figure 6.1. Our fully autonomous prototype, based on a Crazyflie nano-drone.

unmanned aerial vehicles (UAVs), with a form factor of only ∼10 cm in diameter, can become revolutionary *ubiquitous* smart sensing IoT nodes capable of exploring an environment while interacting with it in full autonomy [107], i.e., fulfilling the mission without the need of external resources. The tiny and lightweight design of these pocket-sized drones provides enhanced safety for close-proximity human-machine interactions [19], as well as for indoor scenarios in narrow spaces, such as rescue missions [9].

Tackling such complex use cases requires the drone to reach a high level of intelligence typical of biological systems, which are autonomous and capable of handling multiple concurrent tasks [108] that span from basic control functionality to high-level perception [27, 109]. While autonomous nano-UAVs can already handle multiple basic functionality blocks at run-time (e.g., state estimation, low-level control), they still struggle to pursue multi-objective missions (e.g., safe navigation, exploration, visual recognition), which require computationally intensive multi-tasking perception [9]. In fact, with their ultra-constrained form factor and payload (a few tens of grams), these robots are prevented from hosting high-density/capacity batteries and bulky sensors or processors. This scenario is even further exacerbated by the fraction of the total power budget allotted for the onboard electronics, i.e., 5-10% of the total, which restricts the onboard processors to low-power microcontroller units (MCUs) [29].

We tackle this problem in two steps. First, we enable our drone to tackle a multi-objective mission, i.e., exploration and object detection, by mapping multiple tasks on the two MCUs available aboard our nano-drone. We leverage *i.)* a single-core STM32F405 MCU, with a peak performance <100 M multiply-accumulate (MAC) operations per second, for the execution of lightweight workloads (control, sensor-interfacing tasks), and *ii.)* a parallel ultra-low-power (PULP) multi-core co-processor, GAP8, for coping with convolutional neural network (CNN) workloads (∼1 GMAC/s). In this case, we combine *i*) bio-inspired exploration policies, running on the single-core STM32F405 MCU, and *ii*) a vision-based CNN for object detection aboard a nano-drone running on the GAP8 SoC. We thoroughly characterize four bio-inspired exploration policies by using the ranging measures of multiple single-beam Time-of-Flight (ToF) sensors and run on the STM32F405 MCU. On average, the best exploration strategy covers 83% of a ∼36 m^2 room in a time budget of 3 min. We develop an object detection CNN and deploy it on the GAP8 System-on-Chip (SoC) aboard our nano-drone, analyzing its precision, computational/power costs, and performance by varying its depth. The biggest (deepest) object detector achieves a throughput of 1.6 frame/s and recognizes two classes of objects with a mean average precision (mAP) score of 50% within only 134 mW. When combining the bio-inspired exploration policy with the object detection CNN, the drone recognizes six objects (two classes) with a mean detection rate of 90% (5 independent runs) with a mean flight speed of 0.5 m/s.

Second, we enhance our system by enabling, for the first time, the execution of multiple AI tasks on the GAP8 SoC of the nano-drone. We interleave the CNNs execution of the SoA visual-based CNN for obstacle avoidance, i.e., the Tiny-PULP-Dronet introduced in Chapter 5, with the execution of the object detection CNN introduced in this chapter, achieving a throughput of 1.6 frame/s By doing so, we improve the collision avoidance capabilities of our system, combining the advantages of visual-based collision avoidance and single-beam ToF ranging sensors: the monochrome low-resolution images provide semantic information that we process with the CNN, while ToF sensors provide accurate point-wise distance measurements at close distances (<4 m). We show that this multi-sensory approach for obstacle avoidance can improve up to 60% the collision avoidance capabilities of our nano-drone.

Ultimately, to the best of our knowledge, we present the first fully autonomous nano-drone tackling a multi-objective mission, which consists of exploring, preventing collision, and detecting objects in real time while relying only on onboard sensory and computational resources.

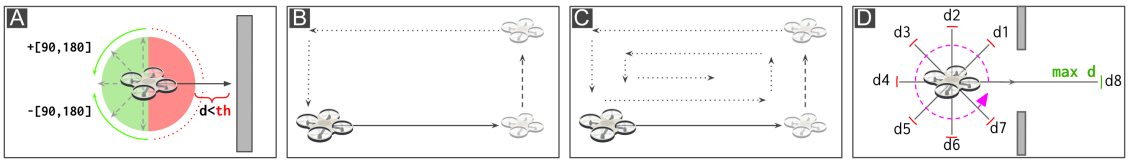


Figure 6.2. Bio-inspired Exploration policies: (A) *Pseudo-random*: inverts the direction by a random angle. (B) *Wall-following*: keeps a fixed distance from the perimeter. (C) *Spiral*: progressively increases the distance from the perimeter. (D) *Rotate-and-measure*: moves along the longest free-space direction.

6.1 System design

6.1.1 Robotic platform

Our robotic platform revolves around the Bitcraze Crazyflie 2.1 nano-drone presented in Chapter 3. This chapter exploits the flight controller PCB, featuring the STM32F405 MCU, to run the state estimation and actuation controls. We also extend the UAV platform with three additional Bitcraze PCBs: the Flow deck, the Multi-ranger deck, and the AI-deck, which features the GAP8 SoC to run the CNN.

6.1.2 Multi-task integration: ToF-based obstacle avoidance and CNN-based.

We exploit our platform to explore an environment while searching for specific objects. To this aim, we separate the problem into three tasks: an *exploration task*, targeting navigation inside an unknown mission area while avoiding collisions, a *visual-based collision avoidance task*, running the Tiny-PULP-Dronet CNN [103], and an *object detection task* that runs the visual object detection SSD algorithm, as described in Sec. 6.1.4.

First, we integrate the exploration task, which is only based on ToF ranging measurements, with the object detection CNN. The exploration task runs a bio-inspired policy to determine the next set-point to feed the flight controller. As detailed in Sec. 6.1.3, we study four different policies, i.e., the *exploration algorithms*, which take as input either the ranging measurements of the ToF sensors or the collision probability CNN output of the Tiny-PULP-Dronet to prevent collisions. On the other side, the performance for the *object detection task* is accounted by measuring the successful detections of target objects present in the mission area. The *exploration* and the *object detection* tasks do not require interaction with each other; therefore, we map them on the two MCUs aboard our nano-drone for concurrent execution. More in detail, we use the two MCUs in a *host-accelerator* configuration. The STM32 is the *host* and takes care of the flight controller and the

exploration policy based on a lightweight state machine. The exploration policies use the ToF measurements from the Multi-ranger deck to determine the next set-point of the drone in terms of forward speed and yaw rate. The Multi-ranger deck acquires raging measurements with a 20 Hz frequency, bounding the exploration policy throughput. On the other hand, GAP8 is the *accelerator* in charge of executing the most computationally demanding CNN algorithm. This task includes camera acquisition and CNN processing and outputs the image-frame coordinates of the objects detected.

Second, we also integrate into our system the visual-based CNN for obstacle avoidance. We adopt a lightweight variant of PULP-Dronet, i.e., Tiny-PULP-Dronet (Chapter 5), featuring a half number of channels in each convolution layers w.r.t. the original version (4), resulting in 83.9 kB of memory footprint when quantized to fixed-point 8-bit precision, achieving a 63 frame/s throughput on GAP8. Its execution on the GAP8 requires additional memory overhead to store intermediate activations, using up to 200 kB of memory budget. Once the inference is done, we forward the output of the CNN, i.e., the visual obstacle avoidance information, to the STM32 and it is used in combination with the ToF ranging measurements: a collision is detected either if the ToF distance measured is < 1 m or if the PULP-Dronet collision output is higher than 0.7. In Section 6.2.4 we will evaluate whether this multi-sensory method, merging the semantic information from low-resolution images and precise distance measurements from ToF sensors, enhances the collision avoidance capabilities of our system.

Ultimately, we enable the execution of both CNNs for visual-based obstacle avoidance and object detection by time-interleaving their execution on GAP8. The processing is done fully onboard, making the final closed-loop system completely autonomous, i.e., with no external communication or computation.

Table 6.1. Mean Average Precision (mAP) of the SSD CNNs trained on Google OpenImages and finetuned on the Himax dataset.

Testing dataset	**Fine-tuning**	**Format**	**SSD size**		
			1×	0.75×	0.5×
OpenImages	*no*	`float32`	59%	47%	43%
Himax	*no*	`float32`	50%	41%	29%
Himax	*yes*	`float32`	55%	46%	43%
Himax	*yes*	`int8`	50%	48%	32%

Figure 6.3. Environments for in-field testing. The test arena is 6.5x5.5 m^2 wide, restricted to 2 ×4.5 m for the narrow-corridor scenario.

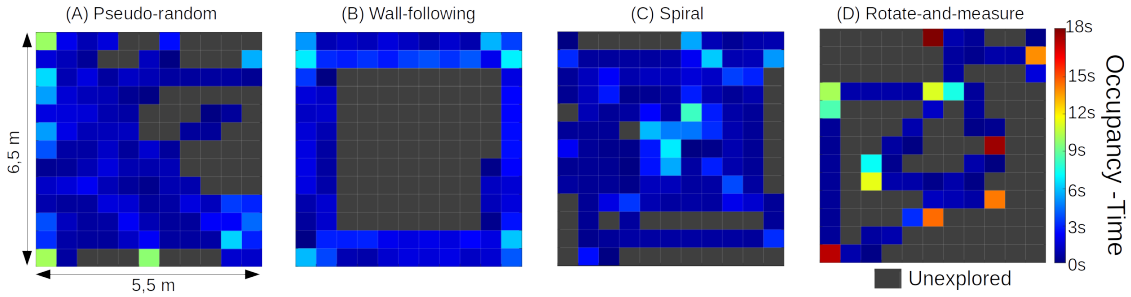


Figure 6.4. Occupancy maps measured using the four exploration policies. The nano-drone flies at 0.5 m/s during the experiments. The color of any cell, which represents a 0.5×0.5 m area, indicates the time spent by the nano-agent in that area (up to 18 sec). Black is used if the cell has not been explored.

6.1.3 Exploration algorithms

Our four bio-inspired exploration algorithms are shown in Fig. 6.2. All the policies rely on the ranging measurements from three ToF sensors: the front, left, and right ones. We assess the performance of these policies by tracking the drone's movements in a 6.5 m×5.5 m room (showed in Figure 6.3-A) equipped with a motion-capture system, tracking at 50 Hz. By discretizing the room's area into cells of 0.5×0.5 m, we plot the heatmap (Fig. 6.4) representing the occupancy time of the drone over a 3 min flight. Our four exploration policies are:

A) Pseudo-random. This approach imitates the pseudo-random movement of biological creatures [75, 109]. The drone flies in a straight line as long as the ToF sensor does not identify obstacles within 1 m. When an obstacle is identified, the drone rotates to a random value, which is always greater than ±90° from the current heading (Fig. 6.2-A) to reduce the likelihood of detecting an obstacle previously identified.

B) Wall-following. This algorithm explores the perimeter of the room following

its walls at a constant distance of 0.5 m, measured by the side ToF sensors (Fig. 6.2-B). The navigation stops when a front obstacle is detected, and resumes after a ±90° turn towards an obstacle-free heading. By construction, this algorithm never explores the inner part of the testing room.

C) Spiral. The drone first explores the environment by performing several concentric perimetral paths (such as wall-following), each with an ever-increasing distance from the external walls (Fig. 6.2-C). Once the room's center has been reached, the process is reversed, i.e., the consecutive perimetral explorations are performed with an ever-decreasing distance to the walls. The process starts over once the drone has reached its initial position. This spiral exploration starts with an initial distance from the walls of 0.5 m, and, for each lap, it is increased/decreased by the same amount.

D) Rotate-and-measure. This algorithm features two sequential phases (Fig. 6.2-D). In the first phase, the drone scans the environment by performing a 360° spin in place while measuring the front distance every 45°. In the second phase, the drone flies towards the most obstacle-free direction for a maximum of 2 m. This policy favors exploring the inner areas of our testing room while frequently neglecting its corners.

6.1.4 Object detection algorithm design

Our object detection pipeline is based on an SSD algorithm composed of a MobilenetV2 feature extractor [58, 110], pre-trained on the full OpenImages [111] dataset, and multiple detection heads, which is the ending part of the CNN that predicts the locations, categories, and confidence scores of objects at different resolution scales. To trade off latency/memory and detection accuracy, we deploy three different CNNs by varying the width multiplier α of the MobilenetV2 backbone. We refer to the different object detection algorithms as *SSD-MbV2-*α, where $\alpha = \{0.5, 0.75, 1.0\}$. The largest *SSD-MbV2-1.0* model features 4.67 M parameters and requires 534 MMAC operations while the *SSD-MbV2-0.75* and *SSD-MbV2-0.5* have, respectively, 2.68 M and 1.34 M parameters and require 358 MMAC and 193 MMAC operations.

We train our SSD models to detect two object categories, *bottles* and *tin cans*, using a subset of images from the OpenImages collection. Given the unbalanced training set, i.e., 1306 and 11306 images for tin cans and bottles, respectively, we balance the dataset by generating additional tin can images through horizontal translation (up to 10% of the image's width). The final split consists of 19142 images for training, 208 for validation, and 663 for testing, where the validation and testing portions match the original dataset splitting. Finally, to overcome the domain shift between the training and real-world data (Fig. 6.5), we add a fine-tuning phase on an additional dataset we collected and called *Himax Dataset*, which includes 321

training images and 279 testing images.

To deploy the object detector on GAP8, we quantize the CNN to 8-bit. We perform an additional fine-tuning step of quantization aware training (QAT) to minimize the mAP loss due to the 8-bit conversion. Then, we exploit the QAT routine included within the Tensorflow Object Detection API framework [110] using symmetric quantization ranges because the software primitives of the target hardware rely on symmetric integer ranges. Lastly, Greenwaves's GAP*flow* toolset[1] is used to produce the C code of the object detection task, constraining the L2 buffer size to 250 kB.

6.2 Results

6.2.1 Object detection evaluation

The SSD models are trained on OpenImages for 1200 epochs using the RMSProp optimizer. We set a learning rate of $8 \cdot 10^{-4}$ with an exponential decay of 0.95 every 24 epochs and a batch size of 24. The OpenImages frames are resized to 320×240 pixels to match the resolution of the drone onboard camera. During training, the images are extended with photometric augmentations, such as flipping, brightness adjustment, random cropping, and grayscale conversion, individually applied with a probability of 0.5. Before deployment, the SSD models are fine-tuned, also applying QAT, on the Himax dataset for 100 epochs, with a learning rate of 10^{-4} retaining the same exponential decay of 0.95 every 10 epochs.

Tab. 6.1 reports the mAP scores (as defined for the COCO dataset [112]) of the

[1]https://greenwaves-technologies.com/gapflow/

Figure 6.5. Google OpenImages sample (left) vs. onboard sensor image (right).

multi-sized SSD models on both the OpenImages and our Himax testing datasets. The largest model achieves an mAP of 59% when trained and tested on images from the OpenImages dataset, up to 16% vs. the smallest model. These scores reduce by 9%, 6%, and 14% for the three models when tested on the Himax dataset. We argue this effect to be due to the onboard camera image quality being lower than the web-retrieved images. After fine-tuning on the Himax training set, the mAP score improves up to 55% for the best model, bridging the accuracy gap for the mAP score on the OpenImages test set. Given the best scores of 50% and 48% after 8-bit quantization, we consider both *SSD-MbV2-1* and *SSD-MbV2-0.75* models for the in-field evaluation.

Tab. 6.2 reports the performance of the SSD CNN running on the GAP8 SoC. We set the operating voltage at 1.2 V and a clock frequency of 160 MHz for the multicore cluster, while the peripheral clock is set to 250 MHz. The largest models (SSD-MbV2-1.0) can process up to 1.6 frame/s, with a computational efficiency of 5.3 *MAC/clock cycles*. Instead, the smaller SSD-MbV2-0.75 and SSD-MbV2-0.5 result, respectively, 1.6× and 2.7× faster than the larger model. The power consumption of the AI-deck reaches a peak of 143.5 mW when running the SSD-MbV2-0.75 model, where the inference task shows the highest compute efficiency, maximizing the memories' bandwidth and processing logic utilization. Conversely, the power consumption decreases to 134.5 mW if running the most accurate SSD-MbV2-1.0 model.

Table 6.2. SSD CNNs' onboard performance.

SSD	Parameters	Operations	Efficiency	Throughput
1×	4.7M	534 MMAC	5.3 MAC/cycles	1.6 FPS
0.75×	2.7M	358 MMAC	5.9 MAC/cycles	2.3 FPS
0.5×	1.2M	193 MMAC	5.3 MAC/cycles	4.3 FPS

6.2.2 Exploration policies evaluation

To assess the performance of our four exploration policies, we calculate the *coverage area* (%) in the obstacle-free testing room described in 6.1.3 (Figure 6.3-A) by calculating the ratio between the visited cells w.r.t. their total (i.e., 143 cells). We mark a cell "visited" when the drone's center of mass falls into it. We evaluate each exploration policy with three average flight speeds (i.e., 0.1 m/s, 0.5 m/s, and 1 m/s), obtaining 12 test configurations. For every configuration, we perform five runs of 3 min each, accounting for a total of 60 runs (3 h flight time).

Fig. 6.6 reports the coverage area of each configuration (policy and speed) averaged on the five runs. The *pseudo-random* and *spiral* policies are those that mainly

benefit from higher speeds, passing from 35% coverage area (at 0.1 m/s) to 74% and 82% (at 0.5 m/s), and finally to 80% and 83% (at 1 m/s), respectively. Instead, the *wall-following* and *rotate-and-measure* exploration policies show a slight improvement passing from a mean flight speed of 0.1 m/s to 0.5 m/s, as much as +17% and +14%, respectively. Contrary to the previous policies, the highest flight speed does not improve the coverage area, i.e., +2% and −1%, respectively. The *wall-following* policy has a limited performance since the drone only explores the room's perimeter (see Fig. 6.4-B). Similarly, the *rotate-and-measure* policy spends the vast majority of the 3 min flight spinning in place and focusing on the center of the room (see Fig. 6.4-D).

6.2.3 In-field closed-loop system evaluation: exploration and object detection

To test our closed-loop system composed of exploration policies, running on the STM32 MCU, and the object detection CNN, running on the GAP8 SoC. To evaluate the in-field performance, we measure the detection rate of each policy/CNN by placing three bottles and three tin cans in the room, relying on the same obstacle-free testing configuration of Sec. 6.2.2: 6.5×5.5 m testing room, 3 min flights. One bottle and one tin can are close to the center, while the other four are near the corners. The final detection rate depends on *i.)* detector's precision and throughput and *ii.)* covered area (partially depending on the flight speed). In Tab. 6.3, we evaluate the best two SSD CNNs (see Sec. 6.2.1) with all four exploration policies and three flight speeds. The bigger *SSD-MbV2-1.0* consistently achieves, for any configuration, an equal or greater detection rate than the medium 0.75× model, suggesting that, in our setup, the detector's accuracy is more important

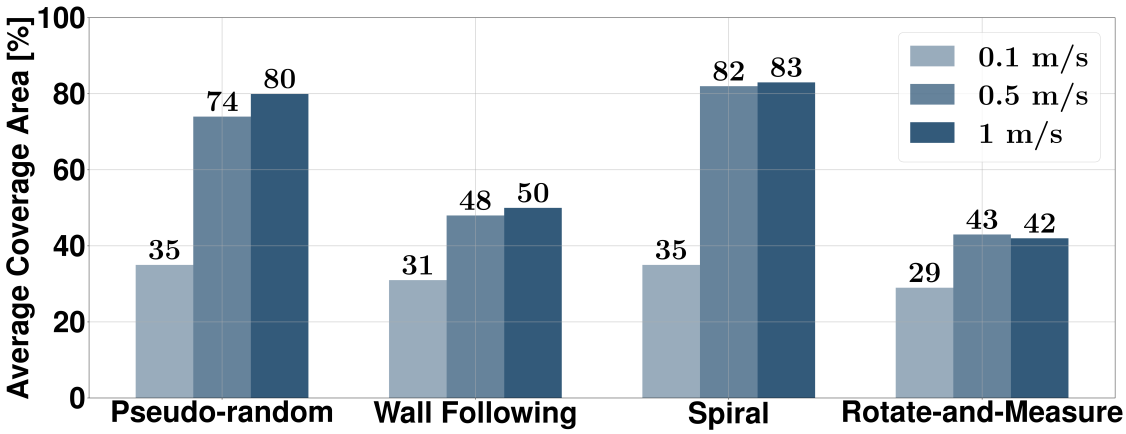


Figure 6.6. Average coverage area (in %) for each exploration policy, varying its mean flight speed (i.e., 0.1 m/s, 0.5 m/s, and 1 m/s).

Table 6.3. Average detection rate: 6 objects, 5 runs of 3 minutes each.

SSD	Flight speed [m/s]	Detection rate			
		Pseudo random	Wall following	Spiral	Rotate and measure
1.0×	0.1	27%	**63%**	67%	**53%**
	0.5	**90%**	50%	**73%**	**53%**
	1	83%	53%	70%	47%
0.75×	0.1	27%	**50%**	33%	47%
	0.5	**80%**	37%	**43%**	**50%**
	1	37%	27%	**43%**	33%

than its throughput (*SSD-MbV2-0.75* has higher throughput than *SSD-MbV2-1.0*, but lower mAP).

Focusing on the *SSD-MbV2-1.0*, the peak performances are achieved by the *pseudo-random* and *spiral* policies, which show the highest detection rate at the intermediate flight speed of 0.5 m/s. In contrast, in Sec. 6.2.2, the best coverage area was obtained by the highest flight speed, which suggests 1 m/s being too high for the limited inference rate of the bigger 1× SSD (i.e., 1.6 frame/s). Then, the *wall-following* and *rotate-and-measure* policies show the lower detection rates, 63% and

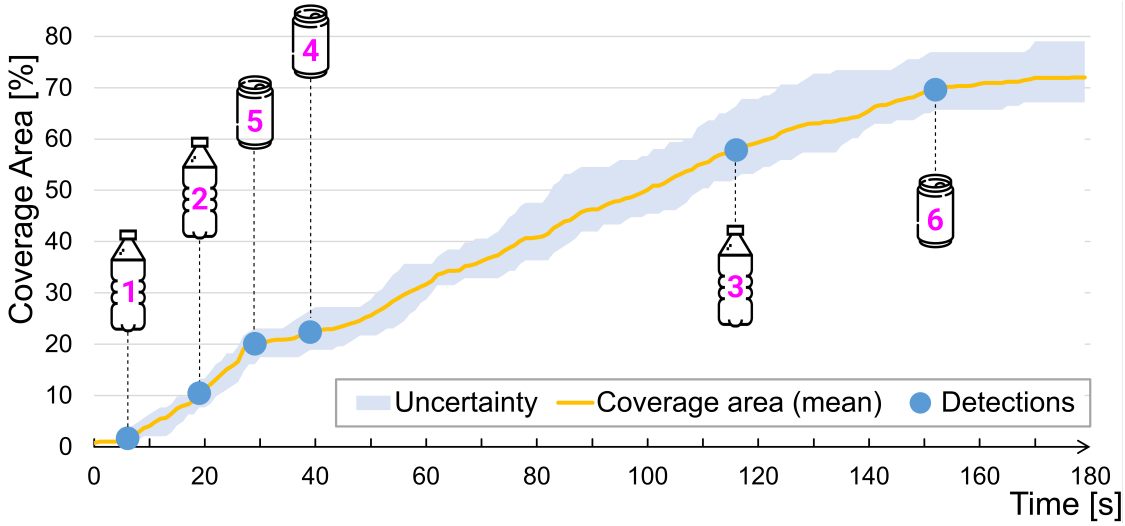


Figure 6.7. Coverage area of the *random* policy over 5 runs (mean and variance). The detection time of the 6 target objects (blue dots) refers to one single run.

Table 6.4. Power breakdown of the robotic platform.

	Motors	CF elect.	AI-deck	Multi-ranger	**Total**
Power [W]	7.32	0.277	0.134	0.286	**8.02**
Percentage	91.31%	3.45%	1.67%	3.57%	**100%**

53% at most, respectively. This behavior is the consequence of the limited coverage area of the policies, where the former misses all the objects placed in the center of the room while the latter marginally explores the perimeter.

In Fig. 6.7, we focus our analysis on the best performing model: *pseudo-random* exploration policy with the nano-drone flying at 0.5 m/s and running the *SSD-MbV2-1.0*. The yellow line shows the coverage area over five 3 min flights, which scores a maximum average of 72% coverage, with a variance of 21%. The blue dots indicate the time each object is recognized for a run achieving 100% detection rate in 154 s.

Lastly, Tab. 6.4 shows the power breakdown of the nano-drone platform, measured by profiling each component individually with the Power Profiler Kit 2 by Nordic Semiconductor. The power cost of the AI-deck, running the biggest *SSD-MbV2-1.0*, accounts for 1.67% of the total power, which is dominated by the motors (91.31%). The other electronics components, i.e., Crazyflie's MCU and the ToF sensors, require the remaining 7.03% of the total power.

6.2.4 Multi-sensory collision avoidance

We test the combination of ToF&PULP-Dronet to assess its effectiveness with respect to systems using a ToF-only collision avoidance method. Fig. 6.3 illustrates three test environments we used: (A) *obstacle-free*, (B) *obstacle-populated* and (C) *narrow-corridor*. In environments B and C, we challenge the nano-drone autonomous explorations using thin obstacles, i.e., chair legs and tripods. Scenario C differs from B as we set up walls to create a 2×4.5 m corridor for the drone flight. In this experiment, a nano-drone takes off from a known position and explores the unknown environment driven by the pseudo-random policy described in Section 6.1.3. For both the ToF-only and ToF&PULP-Dronet settings, we perform 5 and 10 runs for the obstacle-free and narrow corridor environments, respectively, while 20 runs are experimented in the obstacle-populated environment. During all the tests, the flight velocity is set to 0.5 m/s, and the drones fly at 0.3 m altitude. We track the drone trajectories using an external motion capture system, and we manually annotate the crashes with the obstacles.

Table 6.5 shows the results of our in-field experiments, reporting the number of

Table 6.5. In-field obstacle avoidance performance exploration.

Env.	Parameters	SniffyBug	ToF-only	ToF& PULP-Dronet
obstacle free	Crash-free rounds	5 /5	5 /5	-
	Avg. crash/min	0	0	-
	Avg. coverage/min[%]	20.7	22.4	-
obstacle populated	Crash-free rounds	4 /20	3 /20	16 /20
	Avg. crash/min	1.8	1.5	0.2
	Avg. coverage/min[%]	33.2	22.6	10.3
narrow corridor	Crash-free rounds	0 /10	1 /10	7 /10
	Avg. crash/min	2.6	2.4	0.4
	Avg. coverage/min[%]	19.8	45.2	26.7

collision-free missions and the average number of collisions per minute of flight. We also list the average coverage area per minute (expressed in % of the total area), accounted as the number of visited cells w.r.t. the total number of cells (a cell is an area of 5x5 cm^2). In addition to our ToF-only and ToF&PULP-Dronet systems, we benchmark the anti-collision capability of a single drone of the SniffyBug swarm [5] solution, which also relies on the ToF information. In the obstacle-free environment, the ToF solutions already shows perfect performance (100% collision-free), hence we did not run any experiments using PULP-Dronet. Conversely, for the obstacle-populated environment, only 20% or 15% of the trials are successful for, respectively, the SniffyBug and our ToF-only system, dropping to 0% in the narrow corridor for both systems. The poor results are motivated by the missed detection of the narrow obstacles by the ToF sensors. The proposed ToF&PULP-Dronet achieves instead a success rate of 80% and 70% in the two environments. In the narrow-corridor setup, the ToF-only system shows a coverage-per-minute of 45.2%, which is higher than Sniffy-bug and ToF&PULP-Dronet solutions. The latter, in particular, suffers from detecting the corridor with a potential risk of collision. Ultimately, the CNN-based visual collision avoidance improves the collision success rate in cluttered environments from 20% to 80%.

6.2.5 Multi-tasking AI execution

When integrating the PULP-Dronet CNN introduced in Section 6.1.2, and the best object detection CNN (SSD-MbV2-1) evaluated in Section 6.2.3, their interleaved execution results in a throughput of 1.6 frame/s when deployed on GAP8. Table 6.6 summarizes the memory requirements, throughput, and power budget required by the exploration policy, and the two CNNs when running on the STM32 and GAP8 MCUs, respectively. Consequently, the visual-based obstacle avoidance provided

by PULP-Dronet is limited by the throughput SSD-MbV2 CNN, possibly leading to worse performances when compared to the system tested in Section 6.2.4.

Table 6.6. Embedding cost for the multi-task integration. The Parameters are stored in the Flash Memory. Memory refers to the maximum usage of the on-chip L2 memory.

MCU	**Task**	**Throughput** [Hz]	**Parameters** [MB]	**Memory** [KB]	**Power** [mW]
STM32	Exploration policy	20	-	0.1	286
GAP8	SSD	1.7	4.67	250	134
	PULP-Dronet	63	0.08	200	101
	SSD&PULP-Dronet	1.6	4.75	250	133

To study the trade-off between obstacle avoidance and PULP-Dronet's throughput, we record a video dataset of a nano-drone approaching 8 obstacles (6 chairs and 2 tripods) from a distance of 60 cm at a speed of 0.5 m/s. In total, we collected 64 videos at 10 FPS with an average number of 15 frames and at least 12 frames per video, collecting ToF measurements as well. To analyze the impact of multiple throughputs from 1 Hz to 10 Hz, we sub-sample the acquired video, and we apply the PULP-Dronet algorithm. The results in Fig. 6.8 show the number of detected collisions over the dataset. The bottom bars denote the collision assessed only by the ToF (i.e., PULP-Dronet at 0 FPS), which do not vary across the throughput. At $2FPS$, the benefit of PULP-Dronet is bounded to +17% improvement with respect to the ToF-only solution. From $3FPS$ to $6FPS$, the boost in detection rate increases from 34% to 51%, while the performance gain reduces at higher FPS. When increasing the PULP-Dronet throughput of 8 FPS, the overall obstacle

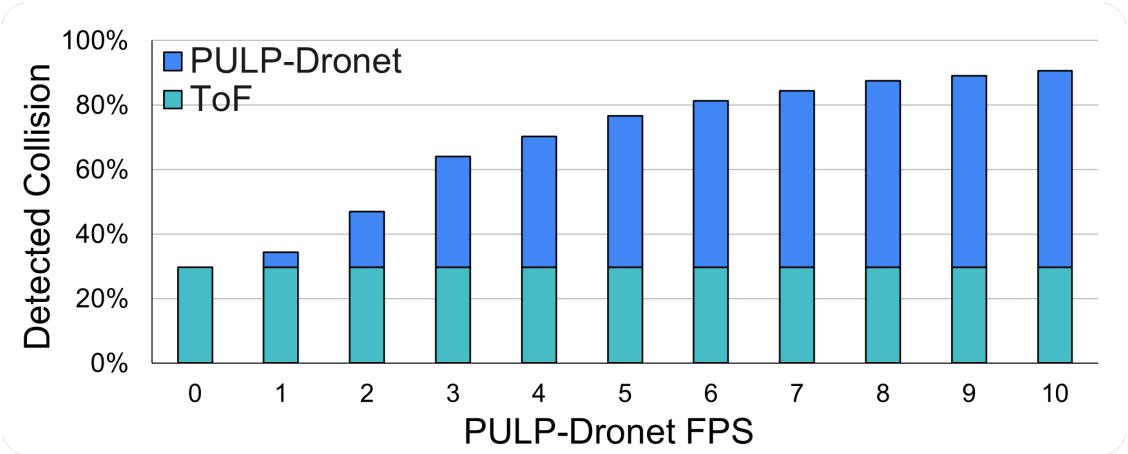


Figure 6.8. PULP-Dronet obstacle detection at different throughput.

detection goes up to 87.5%, peaking at 90.5% at 10 FPS. Overall, we enabled the full onboard execution of two CNNs aboard a nano drone by overcoming the strict memory constraints imposed by MCU-class processors available on nano-UAVs. We took a step forward toward achieving a multi-tasking level of intelligence that is typical of tiny biological systems. However, the integrated execution of two heavy vision-based CNNs is still limited to a throughput of 1.6 FPS. Therefore, this system would further benefit from either a more capable MCU processor or by further optimizing the current bottleneck of the multi-task integration, i.e., the object detection CNN, which takes 573 ms to process.

6.3 Conclusions

This chapter marks a notable achievement for nano-sized UAVs: for the first time, we have successfully enabled a nano-drone to tackle a multi-objective mission autonomously, encompassing the exploration of an unknown environment while avoiding collisions, and searching objects with a vision-based CNN detector. We presented four bio-inspired exploration policies and three versions of object detectors running independently on two resource-limited MCUs aboard the nano-drone. The best configuration reaches a final detection rate of 90%, exploiting *i.)* a pseudo-random policy for exploration, *ii.)* the largest object detection model, and *iii.)* a mean flight speed of 0.5 m/s. This result shows how the higher detection rate can be reached by trading off the detection capabilities of the CNN, its throughput, and the mean flight speed of our nano-drone. Lastly, we enabled for the first time the execution of multiple AI algorithms fully onboard CNNs by time-interleaving the execution of visual-based obstacle avoidance and object detection CNNs on the GAP8 SoC. In particular, we fused both ToF ranging measurements and CNN-based visual obstacle avoidance to improve the reliability of our system when navigating an environment that features thin obstacles. By enabling this AI multi-tasking approach on nano-drones we pave the way for new possibilities in the application of autonomous nano-UAVs.

Chapter 7

Summary and Conclusion

Within the ever-growing field of artificial intelligence, tiny flying robots can revolutionize many aspects of our lives. Once they reach a *true* level of autonomy, i.e., pursuing multiple goals by relying only on their onboard intelligence [13], they can become the perfect tiny robotic helpers. Such a high level of intelligence is akin to insect-sized biological systems, which can seamlessly pursue multiple goals concurrently and with full autonomy [27], but it is still a visionary scenario for miniaturized UAVs.

Enhancing onboard intelligence in UAVs poses significant challenges, as it requires the execution of multiple and complex AI perception workloads. On the one hand, large UAVs, i.e., standard- and micro-sized, cope with this challenge by hosting bulky and powerful processors aboard, which have a power consumption of a few tens of Ws. Such UAVs showcase SoA intelligence capabilities [26, 113, 85, 33], even outperforming humans in some applications [17]. Conversely, tiny drones have stringent payload and size constraints, which restrict their onboard computation devices to ULP processors, challenging the execution of complex AI pipelines fully onboard.

As of today, the forefront of research is constituted by nano-UAVs, i.e., aircraft having about 10 of diameter and a computational power envelope of a few hundreds of mW. Recent progress in the field has led to these small UAVs being capable of successfully executing single CNNs for AI perception entirely onboard. Despite these advancements, nano-UAVs still fall short compared to insect-scale biological systems' capacity to perform complex, multi-objective missions. The challenge lies in their current inability to execute multiple intelligence tasks simultaneously, a capability that is crucial to reach sophisticated skills but remains beyond the reach of these miniaturized UAVs at present.

This thesis demonstrated how to bridge this capability gap, enabling nano-UAVs to

execute multiple perception AI tasks. This achievement was accomplished through three incremental steps. First, in Chapter 4, we showed methodologies and software tools specifically designed to automate the optimization and deployment of CNNs on nano-UAVs, taking into account the stringent computing and memory constraints imposed by their ULP processors. The robustness of this pipeline was field-proven in a challenging competition scenario, winning the first international autonomous nano-drone race.

Second, in Chapter 5, we showed a methodology to minimize the CNN workload on nano-drones. The methodology consists of identifying inactive neurons within the target CNN, followed by architectural modifications to effectively reduce the network's size. When applied to a SoA CNN for visual-based autonomous navigation, we obtained a significantly more compact and computationally efficient network, which is 50× times smaller and 8.5× faster than the original, without any loss in performance metrics.

Third, in Chapter 6, we used our pipeline for automated CNN deployment (Chapter 4) and we leveraged the computational resources freed up with our shrinking methodology (Chapter 5) to successfully enable the execution of multiple AI-based perception tasks aboard our autonomous nano-drone. we enabled the concurrent execution in real-time (1.6 FPS) of a CNN tackling the object detection task and a CNN tackling visual-based navigation and obstacle avoidance.

In conclusion, we took miniaturized flying robots one step closer to the high-level intelligence of tiny biological systems: we demonstrated for the first time that it is possible to overcome both the computational and memory burden imposed by ULP MCUs, allowing the simultaneous execution of multiple AI-based perception tasks on nano-UAVs.

7.1 Outlook and Future Work

This section outlines potential future research aiming to advance the intelligent capabilities of nano-sized UAVs.

7.1.1 Improving AI skills on nano-UAVs

In this thesis, we enabled for the first time the concurrent execution of two real-time AI-based perception tasks on autonomous nano-UAVs, tackling visual-based navigation and object detection. However, to reach a *true* level of autonomy on such robotic platforms, more effort must be put into developing evermore complex AI pipelines. The following are examples of advanced intelligent capabilities that could enhance or be integrated into nano-UAVs.

Autonomous Navigation. State-of-the-art approaches for large UAVs often utilize visual-based simultaneous localization and mapping (SLAM) pipelines for autonomous navigation, creating environmental maps, and planning the trajectory according to it [114, 89]. While essential for enhanced navigation, classical SLAM is computationally too demanding for nano-UAVs [73]. Thus, there is a need for developing lighter monocular SLAM pipelines [115] compatible with MCU constraints, potentially using filtering-based SLAM techniques [116].

Drone Racing. Our current visual-based pipeline enables autonomous navigation and obstacle avoidance in never-seen-before environments. However, to achieve visual-only gate-based navigation, a more sophisticated pipeline encompassing gate detection and trajectory planning is required [35]. Lightweight CNNs for gate pose estimation, proven in larger drones, could also be adapted on the nano-UAVs [34, 33]. Moreover, learning-based methods could revolutionize gate-based navigation on nano-drones by replacing traditional planning, controlling, and perception mechanisms with neural networks. An end-to-end CNN approach could enable nano-drones to navigate gates using raw images for direct UAV control [35].

Exploration and search. We have implemented an object detection CNN capable of identifying two object classes. However, nano-drones must be able to recognize a broader range of objects to tackle practical use cases where exploration and target search are needed, e.g., search and rescue missions. Broadening the object detection skills is essential not only for target identification but also for obstacle avoidance and for landmark recognition to map the environment [117] A promising CNN that could enable this scenario is TinyissimoYOLO [118], which can detect up to 20 object classes with a model size that ranges between 0.58 MB and 3.35 MB, depending on the network's accuracy.

7.1.2 Simulation for developing AI on nano-drones.

The field of nano-drones represents an emerging area of research. as explained in Section 5.2, the scarcity of datasets collected specifically for this platform poses a challenge to developing AI algorithms. We introduced a dataset collector framework specifically tailored for nano-drones to address this issue. However, the manual data collection and labeling process is time-consuming, hindering rapid research and development.

Simulators offer a promising solution to this dataset deficiency in deep learning. There are numerous quad-rotor simulators available, such as Gazebo [119], Hector [120], FlightGoggles [121], Flightmare [122], Nvidia Isaac Gym [123], and Webots [124]. Simulators have been successfully used for various applications in standard-sized and micro-sized UAVs, including drone racing [17, 26, 34, 125], autonomous navigation in unseen-before environments [113, 126], search and rescue missions [127], and environmental monitoring [128]. Leveraging a simulation-based

AI approach, this thesis (Section 4.2) has successfully applied simulation to nano-drones, demonstrating its effectiveness in autonomously navigating a never-seen-before environment. Looking ahead, I envision that the future development of AI on nano-drones will increasingly rely on simulators for data collection. This approach accelerates the development process and offers a versatile and resource-efficient means to refine and test AI algorithms in software.

7.1.3 DNN acceleration and approximate computing.

The current SoA MCU for commercial-off-the-shelf nano-drones is the GAP8 SoC [56], a parallel ultra-low-power processor with a general-purspose 8-core cluster delivering a peak throughput of 5.4 GOps/s [68]. However, newer and more advanced SoCs, such as the GAP9 SoC, have recently emerged. This SoC combines a flexible microcontroller, parallel Digital Signal Processors (DSP), and heterogeneous acceleration capabilities. Notably, the DSP in GAP9 offers a substantial peak throughput of 150.8 GOps/s[1], substantially enhancing both the maximum performance and energy efficiency compared to its predecessor, GAP8. Consequently, it is anticipated that the nano-UAV class will progressively shift towards adopting this advanced SoC. This transition is expected to facilitate the execution of AI pipelines that are more resource-intensive onboard, further expanding the capabilities and applications of these compact aerial vehicles. For example, the use of GAP9 would be able to enhance the simultaneous execution of the *object detection* and *visual obstacle avoidance* CNNs introduced in Chapter 6, now limited at 1.6 FPS

In Chapter 4, we demonstrated the critical role of low-precision integer arithmetic in enabling deep learning inference on tiny, resource-constrained UAVs. This approach significantly speeds up CNN execution, thanks to packed-SIMD instructions [64], and reduces memory usage. Current understanding in the field, supported by studies like [62, 129], suggests that 8-bit quantization can achieve near-zero accuracy degradation, often without retraining. Indeed, our research highlighted the minimal performance impact when reducing data precision from 16-bit to 8-bit during CNN quantization (Section 4.1). To further enhance compression rates with only a modest loss in accuracy, recent research is exploring sub-byte quantization, i.e., using less than 8 bits to quantize DNNs' weights and activations. An area of particular interest is heterogeneous mixed-precision quantization [106], which offers a promising avenue for balancing latency and accuracy trade-offs when quantizing networks with sub-byte precision. Adopting this extreme quantization strategy can unlock the deployment of heavier DL workloads on nano-UAVs, as it reduces CNNs' memory footprint and improves their throughput, ultimately improving energy efficiency when executing DNNs.

[1] `https://greenwaves-technologies.com/low-power-processor/`

Bibliography

[1] V. Niculescu, L. Lamberti, F. Conti, L. Benini, and D. Palossi, "Improving autonomous nano-drones performance via automated end-to-end optimization and deployment of DNNs," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 1–1, 2021.

[2] D. Palossi, A. Marongiu, and L. Benini, "Ultra low-power visual odometry for nano-scale unmanned aerial vehicles," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1647–1650.

[3] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges," *IEEE Access*, vol. 7, pp. 48 572–48 634, 2019.

[4] M. J. Anderson, J. G. Sullivan, J. L. Talley, K. M. Brink, S. B. Fuller, and T. L. Daniel, "The "Smellicopter," a bio-hybrid odor localizing nano air vehicle," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 6077–6082.

[5] B. P. Duisterhof, S. Li, J. Burgu'es, V. J. Reddi, and G. C. de Croon, "Sniffy bug: A fully autonomous swarm of gas-seeking nano quadcopters in cluttered environments," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9099–9106, 2021.

[6] N. Elkunchwar, V. Iyer, M. Anderson, K. Balasubramanian, J. Noe, Y. Talwekar, and S. Fuller, "Bio-Inspired Source Seeking and Obstacle Avoidance on a Palm-Sized Drone," in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, Jun. 2022, pp. 282–289.

[7] S. Awasthi, N. Gramse, C. Reining, and M. Roidl, "UAVs for industries and supply chain management," *ArXiv*, vol. abs/2212.03346, 2022.

[8] N. H. Motlagh, M. Bagaa, and T. Taleb, "UAV-Based IoT Platform: A Crowd Surveillance Use Case," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 128–134, Feb. 2017.

[9] A. Birk, B. Wiggerich, H. Bülow, M. Pfingsthorn, and S. Schwertfeger, "Safety, Security, and Rescue Missions with an Unmanned Aerial Vehicle (UAV)," *Journal of Intelligent & Robotic Systems*, vol. 64, no. 1, pp. 57–76,

Oct. 2011.

[10] U. R. Mogili and BBVL. Deepak, "Review on application of drone systems in precision agriculture," *Procedia computer science*, vol. 133, pp. 502–509, 2018.

[11] K. Z. Y. Ang, X. Dong, W. Liu, G. Qin, S. Lai, K. Wang, D. Wei, S. Zhang, S. K. Phang, X. Chen, M. Lao, Z. Yang, D. Jia, F. Lin, L. Xie, and B. M. Chen, "High-Precision Multi-UAV Teaming for the First Outdoor Night Show in Singapore," *Unmanned Systems*, vol. 06, no. 01, pp. 39–65, Jan. 2018.

[12] R. Bonatti, C. Ho, W. Wang, S. Choudhury, and S. Scherer, "Towards a robust aerial cinematography platform: Localizing and tracking moving targets in unstructured environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 229–236.

[13] T. Smithers, "Autonomy in Robots and Other Agents," *Brain and Cognition*, vol. 34, no. 1, pp. 88–106, Jun. 1997.

[14] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, 2019.

[15] M. Barnell, C. Raymond, S. Smiley, D. Isereau, and D. Brown, "Ultra Low-Power Deep Learning Applications at the Edge with Jetson Orin AGX Hardware," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2022, pp. 1–4.

[16] S. Jung, S. Cho, D. Lee, H. Lee, and D. H. Shim, "A direct visual servoing-based framework for the 2016 IROS Autonomous Drone Racing Challenge," *Journal of Field Robotics*, vol. 35, no. 1, 2018.

[17] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, Aug. 2023.

[18] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64-mW DNN-Based Visual Navigation Engine for Autonomous Nano-Drones," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8357–8371, Oct. 2019.

[19] D. Palossi, N. Zimmerman, A. Burrello, F. Conti, H. Müller, L. M. Gambardella, L. Benini, A. Giusti, and J. Guzzi, "Fully onboard AI-powered human-drone pose estimation on ultra-low power autonomous flying nano-UAVs," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[20] Z. Wei, M. Zhu, N. Zhang, L. Wang, Y. Zou, Z. Meng, H. Wu, and Z. Feng, "UAV-Assisted data collection for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 9, no. 17, pp. 15 460–15 483, 2022.

[21] E. Selinger and D. Durant, "Amazon's Ring: Surveillance as a Slippery Slope Service," *Science as Culture*, vol. 31, no. 1, pp. 92–106, Jan. 2022.

[22] T. Özaslan, S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Inspection of Penstocks and Featureless Tunnel-like Environments Using Micro UAVs,"

in *Field and Service Robotics: Results of the 9th International Conference*, ser. Springer Tracts in Advanced Robotics, L. Mejias, P. Corke, and J. Roberts, Eds. Cham: Springer International Publishing, 2015, pp. 123–136.

[23] A. Fabris, K. Kleber, D. Falanga, and D. Scaramuzza, "Geometry-aware Compensation Scheme for Morphing Drones," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, May 2021, pp. 592–598.

[24] D. Palossi, F. Conti, and L. Benini, "An Open Source and Open Hardware Deep Learning-Powered Visual Navigation Engine for Autonomous Nano-UAVs," in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, May 2019, pp. 604–611.

[25] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, and D. Scaramuzza, "AlphaPilot: Autonomous drone racing," *Autonomous Robots*, vol. 46, no. 1, pp. 307–320, 2022.

[26] C. D. Wagter, F. Paredes-Vallé, N. Sheth, and G. de Croon, "The sensing, state-estimation, and control behind the winning entry to the 2019 Artificial Intelligence Robotic Racing Competition," *Field Robotics*, vol. 2, no. 1, pp. 1263–1290, Mar. 2022.

[27] M. Giurfa and R. Menzel, "Insect visual perception: Complex abilities of simple nervous systems," *Current Opinion in Neurobiology*, vol. 7, no. 4, pp. 505–513, 1997.

[28] S. Ravi, O. Bertrand, T. Siesenop, L.-S. Manz, C. Doussot, A. Fisher, and M. Egelhaaf, "Gap perception in bumblebees," *Journal of Experimental Biology*, vol. 222, no. 2, p. jeb184135, 2019.

[29] R. J. Wood, B. Finio, M. Karpelson, K. Ma, N. O. Pérez-Arancibia, P. S. Sreetharan, H. Tanaka, and J. P. Whitney, "Progress on "Pico" air vehicles," in *Robotics Research : The 15th International Symposium ISRR*, H. I. Christensen and O. Khatib, Eds. Cham: Springer International Publishing, 2017, pp. 3–19.

[30] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield, "Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 4241–4247.

[31] I. Sa, Z. Chen, M. Popović, R. Khanna, F. Liebisch, J. Nieto, and R. Siegwart, "weedNet: Dense semantic weed classification using multispectral images and MAV for smart farming," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 588–595, 2018.

[32] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Lujan, S. Furber, A. J. Davison, P. H. J. Kelly, and M. F. P. O'Boyle, "SLAMBench2: Multi-objective head-to-head benchmarking for visual SLAM," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3637–3644.

[33] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, and

D. Scaramuzza, "Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 690–696.

[34] H. X. Pham, A. Sarabakha, M. Odnoshyvkin, and E. Kayacan, "PencilNet: Zero-Shot Sim-to-Real Transfer Learning for Robust Gate Perception in Autonomous Drone Racing," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 847–11 854, Oct. 2022.

[35] D. Hanover, A. Loquercio, L. Bauersfeld, A. Romero, R. Penicka, Y. Song, G. Cioffi, E. Kaufmann, and D. Scaramuzza, "Autonomous Drone Racing: A Survey," *ArXiv*, vol. abs/2301.01755, 2023.

[36] J. E. Kooi and R. Babuška, "Inclined quadrotor landing using deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 2361–2368.

[37] G. Shi, W. Hönig, X. Shi, Y. Yue, and S.-J. Chung, "Neural-swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions," *IEEE Transactions on Robotics*, pp. 1–17, 2021.

[38] E. Cereda, M. Ferri, D. Mantegazza, N. Zimmerman, L. M. Gambardella, J. Guzzi, A. Giusti, and D. Palossi, "Improving the generalization capability of DNNs for ultra-low power autonomous nano-UAVs," in *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2021, pp. 327–334.

[39] R. J. Bouwmeester, F. Paredes-Vallés, and G. C. H. E. de Croon, "NanoFlowNet: Real-time dense optical flow on a nano quadcopter," *arXiv preprint arXiv:2209.06918*, 2022.

[40] F. Candan, A. Beke, and T. Kumbasar, "Design and deployment of fuzzy PID controllers to the nano quadcopter crazyflie 2.0," in *2018 Innovations in Intelligent Systems and Applications (INISTA)*. IEEE, 2018, pp. 1–6.

[41] O. Dunkley, J. Engel, J. Sturm, and D. Cremers, "Visual-inertial navigation for a camera-equipped 25g nano-quadrotor," in *IROS2014 Aerial Open Source Robotics Workshop*, 2014.

[42] A. Anwar and A. Raychowdhury, "Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes Using Transfer Learning," *IEEE Access*, vol. 8, pp. 26 549–26 560, 2020.

[43] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. J. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[44] W. Zhao, A. Goudar, J. Panerati, and A. P. Schoellig, "Learning-based bias correction for ultra-wideband localization of resource-constrained mobile robots," *CoRR*, vol. abs/2003.09371, 2020.

[45] G. Shi, W. Hönig, Y. Yue, and S.-J. Chung, "Neural-swarm: Decentralized close-proximity multirotor control using learned interactions," in *2020*

IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 3241–3247.

[46] KN. McGuire, C. De Wagter, K. Tuyls, HJ. Kappen, and G. C. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics*, vol. 4, no. 35, 2019.

[47] I. Daramouskas, I. Perikos, I. Hatzilygeroudis, V. J. Lappas, and V. Kostopoulos, "A Methodology For Drones to Learn How to Navigate And Avoid Obstacles Using Decision Trees," in *2020 11th International Conference on Information, Intelligence, Systems and Applications (IISA*, Jul. 2020, pp. 1–4.

[48] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, "Navion: A 2-mW fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, 2019.

[49] Z. Li, Y. Chen, L. Gong, L. Liu, D. Sylvester, D. Blaauw, and H. Kim, "An 879GOPS 243mW 80fps VGA fully visual CNN-SLAM processor for wide-range autonomous exploration," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 134–136.

[50] J.-H. Yoon and A. Raychowdhury, "31.1 a 65nm 8.79TOPS/W 23.82mW mixed-signal oscillator-based NeuroSLAM accelerator for applications in edge robotics," in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, pp. 478–480.

[51] N. K. Manjunath, A. Shiri, M. Hosseini, B. Prakash, N. R. Waytowich, and T. Mohsenin, "An energy efficient EdgeAI autoencoder accelerator for reinforcement learning," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 182–195, 2021.

[52] M. Hosseini and T. Mohsenin, "Binary precision neural network manycore accelerator," *J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 2, Apr. 2021.

[53] S. Bakshi and L. Johnsson, "A highly efficient SGEMM implementation using DMA on the Intel/Movidius myriad-2," in *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2020, pp. 321–328.

[54] L. A. Libutti, F. D. Igual, L. Pinuel, L. De Giusti, and M. Naiouf, "Benchmarking performance and power of USB accelerators for inference with MLPerf," in *2nd Workshop on Accelerated Machine Learning (AccML), Valencia, Spain*, 2020.

[55] F. Conti, D. Palossi, A. Marongiu, D. Rossi, and L. Benini, "Enabling the heterogeneous accelerator model on ultra-low power microcontroller platforms," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 1201–1206.

[56] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "GAP-8: A RISC-V SoC for AI at the edge of the IoT," in *2018 IEEE 29th International Conference on Application-Specific Systems, Architectures and*

Processors (ASAP), 2018, pp. 1–4.

[57] E. Cereda, L. Crupi, M. Risso, A. Burrello, L. Benini, A. Giusti, D. Jahier Pagliari, and D. Palossi, "Deep neural network architecture search for accurate visual pose estimation aboard nano-UAVs," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 6065–6071.

[58] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," 2019.

[59] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "DORY: Automatic end-to-end deployment of real-world DNNs on low-cost IoT MCUs," *IEEE Transactions on Computers*, pp. 1–1, 2021.

[60] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers," in *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds., vol. 2, 2020, pp. 326–335.

[61] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for MobileNetV3," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1314–1324.

[62] J. Choi, S. Venkataramani, V. V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, "Accurate and efficient 2-bit quantized neural networks," in *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia, Eds., vol. 1, 2019, pp. 348–359.

[63] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[64] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[65] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden, "TensorFlow lite micro: Embedded machine learning on TinyML systems," *arXiv preprint arXiv:2010.08678*, 2021.

[66] L. Geiger and P. Team, "Larq: An open-source library for training binarized neural networks," *Journal of Open Source Software*, vol. 5, no. 45, p. 1746, 2020.

[67] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for arm cortex-m CPUs," *arXiv preprint arXiv:1801.06601*, Jan. 2018.

[68] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, no. 2164, p. 20190155, 2020.

[69] J. Dupeyroux, RAOUL. DINAUX, NIKHIL. WESSENDORP, and G. De Croon, "A novel obstacle detection and avoidance dataset for drones," in *System Engineering for Constrained Embedded Systems*, ser. DroneSE and RAPIDO. New York, NY, USA: Association for Computing Machinery, 2022, pp. 8–13.

[70] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, 2018.

[71] Y. Chang, Y. Cheng, J. Murray, S. Huang, and G. Shi, "The HDIN dataset: A real-world indoor UAV dataset with multi-task labels for visual-based navigation," *Drones*, vol. 6, no. 202, 2022.

[72] M. Navardi, E. Humes, and T. Mohsenin, "E2EdgeAI: Energy-efficient edge computing for deployment of vision-based DNNs on autonomous tiny drones," in *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, 2022, pp. 504–509.

[73] D. Palossi, F. Tombari, S. Salti, M. Ruggiero, L. Stefano, and L. Benini, "Gpu-shot: Parallel optimization for real-time 3d local description," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2013, pp. 584–591.

[74] K. McGuire, G. de Croon, and K. Tuyls, "A comparative study of bug algorithms for robot navigation," *Robotics and Autonomous Systems*, vol. 121, p. 103261, 2019.

[75] Q.-l. Xu, "Randombug: Novel path planning algorithm in unknown environment," *The Open Electrical & Electronic Engineering Journal*, vol. 8, no. 1, 2014.

[76] E. Magid and E. Rivlin, "CautiousBug: A competitive algorithm for sensory-based robot navigation," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2757–2762 vol.3.

[77] Q.-L. Xu and G.-Y. Tang, "Vectorization path planning for autonomous mobile agent in unknown environment," *Neural Computing and Applications*, vol. 23, no. 7, pp. 2129–2135, 2013.

[78] I. Kamon, E. Rivlin, and E. Rimon, "A new range-sensor based globally convergent navigation algorithm for mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1. IEEE, 1996, pp. 429–435.

[79] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Computer Vision – ECCV*

2016, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 21–37.

[80] T. Q. Khoi, N. A. Quang, and N. K. Hieu, "Object detection for drones on Raspberry Pi potentials and challenges," *IOP Conference Series: Materials Science and Engineering*, vol. 1109, no. 1, p. 012033, Mar. 2021.

[81] L. Lamberti, M. Rusci, M. Fariselli, F. Paci, and L. Benini, "Low-Power License Plate Detection and Recognition on a RISC-V Multi-Core MCU-Based Vision System," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2021, pp. 1–5.

[82] H. Zhou, Z. Hu, S. Liu, and S. Khan, "Efficient 2D Graph SLAM for Sparse Sensing," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 6404–6411.

[83] O. Dunkley, J. J. Engel, J. Sturm, and D. Cremers, "Visual-Inertial Navigation for a Camera-Equipped 25g Nano-Quadrotor," in *Proc. IROS Aerial Open Source Robot. Workshop*, 2014, pp. 1–2.

[84] K. McGuire, G. de Croon, C. De Wagter, K. Tuyls, and H. Kappen, "Efficient Optical Flow and Stereo Vision for Velocity Estimation and Obstacle Avoidance on an Autonomous Pocket Drone," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1070–1076, Apr. 2017.

[85] P. Foehn, A. Romero, and D. Scaramuzza, "Time-Optimal Planning for Quadrotor Waypoint Flight," *Science Robotics*, vol. 6, no. 56, p. eabh1221, Jul. 2021.

[86] S. Li, E. van der Horst, P. Duernay, C. De Wagter, and G. C. H. E. de Croon, "Visual Model-Predictive Localization for Computationally Efficient Autonomous Racing of a 72-g Drone," *Journal of Field Robotics*, vol. 37, no. 4, pp. 667–692, 2020.

[87] D. Palossi, J. Singh, M. Magno, and L. Benini, "Target Following on Nano-Scale Unmanned Aerial Vehicles," in *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, Jun. 2017, pp. 170–175.

[88] A. Gomez, M. Magno, M. F. Lagadec, and L. Benini, "Precise, energy-efficient data acquisition architecture for monitoring radioactivity using self-sustainable wireless sensor nodes," *IEEE Sensors Journal*, vol. 18, no. 1, pp. 459–469, 2017.

[89] G. Loianno, D. Scaramuzza, and V. Kumar, "Special issue on high-speed vision-based autonomous navigation of UAVs," *Journal of Field Robotics*, vol. 35, no. 1, pp. 3–4, 2018.

[90] *Analog Devices MAX78000: Artificial Intelligence Microcontroller with UltraLow-Power Convolutional Neural Network Accelerator*, May 2021.

[91] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[92] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele, "RocketLogger: Mobile power logger for prototyping IoT devices: Demo abstract," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, ser. SenSys '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 288–289.

[93] B. Sun and K. Saenko, "Deep coral: Correlation alignment for deep domain adaptation," in *European Conference on Computer Vision*. Springer, 2016, pp. 443–450.

[94] H. Ren, D. Anicic, and T. Runkler, "TinyOL: TinyML with online-learning on microcontrollers," *arXiv preprint arXiv:2103.08295*, 2021.

[95] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.

[96] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*. MIT press, 2011.

[97] H. Müller, D. Palossi, S. Mach, F. Conti, and L. Benini, "Fünfiiber-drone: A modular open-platform 18-grams autonomous nano-drone," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1610–1615.

[98] L. Bigazzi, M. Basso, S. Gherardini, and G. Innocenti, "Mitigating latency problems in vision-based autonomous UAVs," in *2021 29th Mediterranean Conference on Control and Automation (MED)*, 2021, pp. 1203–1208.

[99] V. Niculescu, L. Lamberti, D. Palossi, and L. Benini, "Automated Tuning of End-to-end Neural Flight Controllers for Autonomous Nano-drones," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Jun. 2021, pp. 1–4.

[100] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks," *Journal of Machine Learning Research*, vol. 22, no. 241, pp. 1–124, Sep. 2021.

[101] N. Hossein Motlagh, T. Taleb, and O. Arouk, "Low-altitude unmanned aerial vehicles-based internet of things services: Comprehensive survey and future perspectives," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 899–922, 2016.

[102] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges," *IEEE access : practical innovations, open solutions*, vol. 7, pp. 48 572–48 634, 2019.

[103] L. Lamberti, V. Niculescu, M. Barciś, L. Bellone, E. Natalizio, L. Benini, and D. Palossi, "Tiny-PULP-Dronets: Squeezing Neural Networks for Faster and Lighter Inference on Multi-Tasking Autonomous Nano-Drones," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems*

(AICAS), Jun. 2022, pp. 287–290.

[104] F. Conti, "Technical report: NEMO DNN quantization for deployment model," *arXiv preprint arXiv:2004.05930*, 2020.

[105] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017.

[106] A. Capotondi, M. Rusci, M. Fariselli, and L. Benini, "CMix-NN: Mixed low-precision CNN library for memory-constrained edge devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 871–875, 2020.

[107] D. Palossi, A. Gomez, S. Draskovic, K. Keller, L. Benini, and L. Thiele, "Self-sustainability in nano unmanned aerial vehicles: A blimp case study," in *Proceedings of the Computing Frontiers Conference*, 2017.

[108] M. V. Srinivasan, "Honeybees as a model for the study of visually guided flight, navigation, and biologically inspired robotics," *Physiological Reviews*, vol. 91, no. 2, pp. 413–460, 2011.

[109] B. Webb, "The internal maps of insects," *Journal of Experimental Biology*, vol. 222, 2019.

[110] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/Accuracy trade-offs for modern convolutional object detectors," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[111] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov *et al.*, "The open images dataset v4," *International Journal of Computer Vision*, vol. 128, no. 7, pp. 1956–1981, 2020.

[112] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European Conference on Computer Vision*. Springer International Publishing, 2014, pp. 740–755.

[113] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and Davide Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021.

[114] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.

[115] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.

[116] V. Niculescu, T. Polonelli, M. Magno, and L. Benini, "NanoSLAM: Enabling Fully Onboard SLAM for Tiny Robots," *IEEE Internet of Things Journal*,

pp. 1–1, 2023.

[117] N. Hughes, Y. Chang, and L. Carlone, "Hydra: A Real-time Spatial Perception System for 3D Scene Graph Construction and Optimization," *Robotics: Science and Systems XVIII*, Jun. 2022.

[118] J. Moosmann, H. Mueller, N. Zimmerman, G. Rutishauser, L. Benini, and M. Magno, "Flexible and Fully Quantized Ultra-Lightweight TinyissimoYOLO for Ultra-Low-Power Edge Systems," Jul. 2023.

[119] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2149–2154, 2004.

[120] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, U. Klingauf, and O. von Stryk, "Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots," in *RoboCup 2013: Robot World Cup XVII*, ser. Lecture Notes in Computer Science, S. Behnke, M. Veloso, A. Visser, and R. Xiong, Eds. Berlin, Heidelberg: Springer, 2014, pp. 624–631.

[121] W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, "FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Macau, China: IEEE Press, Nov. 2019, pp. 6941–6948.

[122] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Conference on Robot Learning*, 2020.

[123] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning," Aug. 2021.

[124] O. Michel, "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004.

[125] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep Drone Racing: From Simulation to Reality With Domain Randomization," *IEEE Transactions on Robotics*, vol. 36, no. 1, pp. 1–14, Feb. 2020.

[126] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 3948–3955.

[127] J. G. C. Zuluaga, J. P. Leidig, C. Trefftz, and G. Wolffe, "Deep Reinforcement Learning for Autonomous Search and Rescue," in *NAECON 2018 - IEEE National Aerospace and Electronics Conference*, Jul. 2018, pp. 521–524.

[128] W. J. Yun, S. Park, J. Kim, M. Shin, S. Jung, D. A. Mohaisen, and J.-H. Kim, "Cooperative Multiagent Deep Reinforcement Learning for Reliable

Surveillance via Autonomous Multi-UAV Control," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 10, pp. 7086–7096, Oct. 2022.

[129] M. Nagel, M. V. Baalen, T. Blankevoort, and M. Welling, "Data-Free Quantization Through Weight Equalization and Bias Correction," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. Seoul, Korea (South): IEEE, Oct. 2019, pp. 1325–1334.